

Logiche specifiche per la sicurezza informatica

Parte I: Fondamenti e Contesto Storico

Introduzione al Problema della Formalizzazione della Sicurezza

La sicurezza informatica ha rappresentato, fin dalle sue origini, una sfida peculiare per la formalizzazione matematica e logica. A differenza di molti altri ambiti dell'informatica, dove la correttezza di un algoritmo o di un sistema può essere verificata attraverso l'analisi del suo comportamento in condizioni normali, la sicurezza richiede di considerare non solo il funzionamento previsto ma anche tutti i possibili comportamenti malevoli di un avversario. Questo aspetto introduce una dimensione epistemica e modale nella trattazione formale, poiché diventa necessario ragionare su ciò che un agente può conoscere, dedurre, o fare in presenza di informazioni parziali.

Le prime formalizzazioni della crittografia e dei protocolli di sicurezza risalgono agli anni Settanta e Ottanta del ventesimo secolo, quando i ricercatori cominciarono a rendersi conto che l'approccio informale alla progettazione di protocolli crittografici portava sistematicamente a vulnerabilità impreviste. Un protocollo che sembrava sicuro a un'analisi superficiale poteva rivelarsi vulnerabile ad attacchi sofisticati che sfruttavano sottili interazioni tra le diverse componenti del sistema. La necessità di strumenti formali divenne quindi imperativa, non solo per verificare la correttezza dei protocolli esistenti ma anche per guidare la progettazione di nuovi sistemi sicuri.

Il contesto storico in cui si svilupparono le logiche per la sicurezza è profondamente legato all'evoluzione della crittografia moderna. Con l'introduzione della crittografia a chiave pubblica da parte di Diffie e Hellman nel 1976, seguita dal celebre sistema RSA di Rivest, Shamir e Adleman nel 1978, divenne chiaro che i protocolli crittografici non erano semplici applicazioni di primitive matematiche, ma sistemi complessi in cui la composizione di operazioni sicure poteva produrre risultati insicuri. Questa realizzazione portò alla nascita di un'intera area di ricerca dedicata alla formalizzazione e verifica dei protocolli di sicurezza.

Le Due Tradizioni nella Formalizzazione della Sicurezza

Nella storia dello sviluppo delle logiche per la sicurezza, si possono identificare due tradizioni principali che hanno seguito approcci fondamentalmente diversi al problema della formalizzazione. Queste due tradizioni, che potremmo chiamare l'approccio simbolico o astratto e l'approccio computazionale o concreto, hanno coesistito per decenni, ciascuna con i propri vantaggi e limitazioni, prima che si sviluppasse un corpus di risultati capaci di collegare i due paradigmi.

L'approccio simbolico tratta le operazioni crittografiche come funzioni perfette e astratte. In questo modello, la cifratura è vista come una black box che trasforma un messaggio in un

testo cifrato, e l'unica maniera per ottenere il messaggio originale è possedere la chiave corretta. Non esistono attacchi crittanalitici, non esistono debolezze computazionali, non esistono probabilità di successo negli attacchi. Un messaggio cifrato con una chiave è completamente opaco a chiunque non possieda quella chiave. Questo livello di astrazione permette di concentrarsi sulla struttura logica del protocollo, sulle dipendenze tra messaggi, sul flusso di informazioni e sulla conoscenza degli agenti, senza perdersi nei dettagli implementativi o nelle complessità della teoria della complessità computazionale.

L'approccio computazionale, al contrario, modella le operazioni crittografiche come algoritmi concreti eseguiti su macchine di Turing probabilistiche con risorse limitate. In questo contesto, la sicurezza non è assoluta ma computazionale: un sistema è sicuro se ogni avversario con risorse polinomiali ha solo una probabilità trascurabile di violare una proprietà di sicurezza. Questo approccio, pur essendo più realistico e aderente alla pratica crittografica moderna, introduce una complessità notevole nelle dimostrazioni e rende difficile l'automazione della verifica.

Per molti anni, queste due tradizioni sono rimaste separate. I ricercatori nell'area della verifica formale dei protocolli preferivano l'approccio simbolico per la sua trattabilità e la possibilità di automatizzare le verifiche, mentre i crittografi rigorosi consideravano tale approccio troppo idealizzato per fornire garanzie significative sulla sicurezza reale dei sistemi. Il ponte tra questi due mondi è stato costruito gradualmente attraverso una serie di risultati di computational soundness, che mostrano sotto quali condizioni una dimostrazione di sicurezza nel modello simbolico implica sicurezza nel modello computazionale. Questi risultati rappresentano uno degli sviluppi più importanti nella teoria della sicurezza informatica degli ultimi vent'anni.

I Fondamenti Epistemici della Sicurezza

Una delle caratteristiche distintive delle logiche per la sicurezza è il loro carattere intrinsecamente epistemico. Quando ragioniamo sulla sicurezza di un protocollo, non ci limitiamo a descrivere ciò che accade nel sistema, ma dobbiamo anche modellare ciò che i vari agenti coinvolti fanno, credono, o possono dedurre. Questa dimensione epistemica è fondamentale perché la sicurezza stessa è una proprietà che riguarda la conoscenza: un sistema è sicuro rispetto a una certa proprietà se un avversario non può venire a conoscenza di informazioni che dovrebbero rimanere segrete, non può convincere un altro agente di fatti falsi, non può impersonare altri agenti, e così via.

La logica epistemica standard, sviluppata principalmente nell'ambito della filosofia e dell'intelligenza artificiale, fornisce un punto di partenza naturale per questa formalizzazione. In una logica epistemica, si introducono operatori modali della forma $K_i \varphi$, che si legge "l'agente i sa che φ è vero". Questi operatori soddisfano tipicamente gli assiomi del sistema logico S5, che cattura l'idea di conoscenza come credenza vera giustificata in un contesto di informazione perfetta. Gli assiomi principali sono: la distribuzione della conoscenza sulle implicazioni logiche, la veridicità della conoscenza (se un agente sa qualcosa, allora quella cosa è vera), l'introspezione positiva (se un agente sa qualcosa, sa di saperlo), e l'introspezione negativa (se un agente non sa qualcosa, sa di non saperlo).

Tuttavia, l'applicazione diretta della logica epistemica standard ai protocolli di sicurezza si scontra con diverse difficoltà. Innanzitutto, nei protocolli crittografici la conoscenza non è assoluta ma relativa alle capacità computazionali degli agenti. Un agente può "conoscere" un testo cifrato senza conoscere il messaggio in chiaro, anche se matematicamente il messaggio è determinato dal testo cifrato e dalla chiave. Questa distinzione tra possesso di un dato e capacità di estrarre informazione da quel dato richiede una nozione più sfumata di conoscenza.

In secondo luogo, la conoscenza nei protocolli di sicurezza è spesso derivata attraverso operazioni crittografiche. Un agente può acquisire conoscenza di un messaggio non solo ricevendolo direttamente, ma anche decifrando un testo cifrato con una chiave che possiede, verificando una firma digitale, calcolando un hash, o componendo informazioni parziali. La logica epistemica deve quindi essere arricchita con regole di inferenza che riflettono le capacità computazionali degli agenti e le proprietà delle primitive crittografiche.

In terzo luogo, nei protocolli di sicurezza si manifesta spesso una forma di conoscenza comune o distribuita tra gli agenti. La nozione di conoscenza comune, introdotta nel contesto della teoria dei giochi e dei sistemi distribuiti, diventa cruciale quando si vuole garantire che tutti gli agenti coinvolti in un protocollo concordino su determinati fatti, come l'autenticità di una chiave o il completamento di una fase del protocollo. La formalizzazione della conoscenza comune richiede operatori modali più sofisticati e introduce questioni tecniche delicate relative alla loro semantica e assiomatizzazione.

La Temporalità nei Protocolli di Sicurezza

Oltre alla dimensione epistemica, i protocolli di sicurezza hanno una dimensione temporale essenziale. Un protocollo non è un'istantanea statica di uno stato del sistema, ma una sequenza di azioni che si svolgono nel tempo, con vincoli su quale agente può compiere quale azione in quale momento, e con dipendenze causali tra le diverse fasi. La formalizzazione di questi aspetti temporali richiede l'uso di logiche temporali, che permettono di esprimere proprietà come "dopo l'invio di un messaggio, eventualmente verrà ricevuta una risposta" o "una chiave rimane segreta fino a quando non viene compromessa".

Le logiche temporali più utilizzate nel contesto della sicurezza sono le varianti della logica temporale lineare (LTL) e della logica temporale con alberi computazionali (CTL). In LTL, il tempo è visto come una sequenza lineare di stati, e si possono esprimere proprietà usando operatori come X (next, nel prossimo stato), F (finally, eventualmente in futuro), G (globally, sempre in futuro), e U (until, fino a quando). In CTL, invece, si considera un albero di possibili evoluzioni future, ramificato nei punti in cui il sistema può fare scelte non deterministiche, e gli operatori temporali sono preceduti da quantificatori sui cammini, come A (per tutti i cammini) ed E (esiste un cammino).

La combinazione di operatori epistemici e temporali dà origine a logiche epistemo-temporali, in cui si possono esprimere proprietà sofisticate come "l'agente A saprà eventualmente che la proprietà ϕ è sempre stata vera" o "esiste un momento futuro in cui tutti gli agenti sapranno di avere una conoscenza comune su un certo fatto". Queste logiche, pur essendo molto espressive, presentano sfide tecniche considerevoli in termini di

decidibilità, complessità computazionale della verifica, e sviluppo di strumenti automatici di analisi.

La Composizionalità e i Process Calculi

Un'altra dimensione fondamentale nella formalizzazione dei protocolli di sicurezza è la composizionalità. I sistemi di sicurezza moderni non sono monolitici ma composti da molteplici agenti che eseguono concorrentemente diverse istanze dello stesso protocollo o di protocolli diversi, comunicando attraverso canali potenzialmente controllati da un avversario. La formalizzazione di questi sistemi richiede un framework che permetta di descrivere il comportamento di processi concorrenti e la loro composizione.

I process calculi, sviluppati nell'ambito della teoria della concorrenza, forniscono proprio questo tipo di framework. Il π -calculus, introdotto da Milner, Parrow e Walker negli anni Novanta, è particolarmente adatto alla modellazione di sistemi con topologie di comunicazione dinamiche, dove i canali di comunicazione stessi possono essere trasmessi come dati. Questa caratteristica lo rende ideale per modellare protocolli crittografici, dove le chiavi (che possono essere viste come nomi di canali privati) vengono generate, distribuite e utilizzate in modi complessi.

Nel contesto della sicurezza, il π -calculus è stato esteso in diverse direzioni per incorporare primitive crittografiche e nozioni di segretezza. Il spi-calculus di Abadi e Gordon aggiunge al π -calculus operazioni di cifratura e decifratura, insieme a un'analisi statica della segretezza basata sulla nozione di equivalenza osservazionale. L'idea è che due processi sono equivalenti se un avversario esterno non può distinguerli osservando le loro interazioni con l'ambiente. Questa nozione di equivalenza cattura molte proprietà di sicurezza interessanti, come la segretezza di dati sensibili e l'anonimato degli agenti.

L'applied π -calculus, sviluppato successivamente da Abadi e Fournet, generalizza ulteriormente questo approccio permettendo di definire teorie equazionali arbitrarie per le primitive crittografiche. Invece di avere operazioni di cifratura e decifratura hard-coded nel calcolo, si specifica una segnatura di funzioni (che rappresentano le operazioni crittografiche) insieme a equazioni che descrivono le loro proprietà algebriche. Ad esempio, si può specificare che la decifratura di un testo cifrato con la chiave giusta restituisce il messaggio originale attraverso l'equazione $\text{dec}(\text{enc}(m, k), k) = m$. Questo approccio basato su teorie equazionali è estremamente flessibile e ha permesso di modellare un'ampia varietà di primitive crittografiche, dai sistemi simmetrici a quelli asimmetrici, dalle funzioni hash alle firme digitali, fino a costruzioni avanzate come la cifratura omomorfa e le prove a conoscenza zero.

Le Proprietà di Sicurezza e la loro Classificazione

Prima di addentrarci nelle specifiche logiche utilizzate per la verifica della sicurezza, è importante chiarire quali sono le proprietà di sicurezza che vogliamo verificare. La letteratura sulla sicurezza informatica ha sviluppato una ricca tassonomia di proprietà di sicurezza, che possiamo raggruppare in alcune categorie principali.

La segretezza, o confidenzialità, è forse la proprietà più intuitiva: certi dati devono rimanere privati e non devono essere accessibili ad agenti non autorizzati. Tuttavia, anche questa nozione apparentemente semplice si presenta in diverse varianti. La segretezza forte richiede che un avversario non possa ottenere alcuna informazione sul dato segreto. La segretezza debole permette che l'avversario apprenda alcune informazioni parziali, purché non sufficienti a ricostruire completamente il segreto. La segretezza forward garantisce che la compromissione di chiavi future non permetta di decifrare comunicazioni passate. La segretezza backward garantisce il viceversa.

L'autenticazione riguarda la capacità di verificare l'identità degli agenti coinvolti in un protocollo. Anche qui esistono diverse forme. L'autenticazione unilaterale richiede che solo una delle parti verifichi l'identità dell'altra. L'autenticazione mutua richiede che entrambe le parti verifichino reciprocamente le loro identità. L'autenticazione debole garantisce semplicemente che un agente stia comunicando con qualcuno che possiede le credenziali appropriate. L'autenticazione forte, o entity authentication, garantisce che l'agente con cui si comunica sia effettivamente presente e stia partecipando alla sessione corrente del protocollo, non semplicemente riproducendo messaggi di sessioni precedenti.

L'integrità concerne la protezione dei dati da modifiche non autorizzate. Un messaggio ha integrità se il ricevente può verificare che non è stato alterato durante la trasmissione. Questa proprietà è tipicamente garantita attraverso funzioni hash crittografiche o codici di autenticazione dei messaggi (MAC). L'integrità si declina anch'essa in diverse forme, a seconda che si voglia proteggere singoli messaggi o intere sequenze, e a seconda delle garanzie fornite contro la rimozione, l'inserimento o il riordino di messaggi.

Il non ripudio è una proprietà che garantisce che un agente che ha inviato un messaggio non possa successivamente negare di averlo fatto. Questa proprietà è tipicamente ottenuta attraverso firme digitali che solo il mittente può produrre ma che chiunque può verificare. Il non ripudio è cruciale in applicazioni come il commercio elettronico e i contratti digitali, dove è necessario avere prove verificabili delle azioni compiute dagli agenti.

L'anonimato e le sue varianti (unlinkability, unobservability) riguardano la protezione dell'identità e delle azioni degli agenti. Un protocollo garantisce anonimato se un osservatore esterno non può determinare quale agente ha compiuto quale azione. L'unlinkability richiede che un osservatore non possa determinare se due azioni osservate sono state compiute dallo stesso agente o da agenti diversi. L'unobservability richiede che un osservatore non possa nemmeno determinare se un agente sta partecipando o meno al protocollo.

Oltre a queste proprietà fondamentali, esistono molte altre proprietà di sicurezza più specifiche, come la fairness nei protocolli di scambio (garantire che nessuna parte possa ottenere un vantaggio ingiusto), la receipt-freeness nelle votazioni elettroniche (impedire che un votante possa provare come ha votato), la coercion-resistance (protezione contro la coercizione), e molte altre proprietà specifiche di particolari domini applicativi.

Una distinzione importante nella classificazione delle proprietà di sicurezza è quella tra proprietà di traccia e proprietà di equivalenza. Le proprietà di traccia sono quelle che possono essere verificate osservando singole esecuzioni del protocollo: ad esempio, "il dato segreto non viene mai inviato in chiaro" è una proprietà di traccia. Le proprietà di equivalenza richiedono invece di confrontare il comportamento del sistema in scenari

diversi: ad esempio, l'anonimato può essere formalizzato dicendo che un osservatore non può distinguere un'esecuzione del protocollo con l'agente A da un'esecuzione con l'agente B. Le proprietà di equivalenza sono generalmente più difficili da verificare delle proprietà di traccia, e richiedono tecniche formali più sofisticate.

La BAN Logic: Pioniera delle Logiche per l'Autenticazione

Genesi e Motivazione della BAN Logic

La BAN logic, così chiamata dalle iniziali dei suoi creatori Michael Burrows, Martin Abadi e Roger Needham, rappresenta uno dei primi e più influenti tentativi di creare un framework logico formale per l'analisi dei protocolli di autenticazione. Pubblicata nel 1989 in un articolo fondamentale intitolato "A Logic of Authentication", questa logica nacque dall'osservazione che molti protocolli di autenticazione, anche quelli progettati da esperti e ampiamente utilizzati in pratica, contenevano vulnerabilità sottili che emergevano solo dopo anni di utilizzo.

Il contesto storico è importante per comprendere la genesi della BAN logic. Alla fine degli anni Ottanta, protocolli come Needham-Schroeder, progettato per la distribuzione di chiavi in sistemi distribuiti, erano stati analizzati informalmente per anni e considerati sicuri dalla comunità. Tuttavia, proprio in quel periodo, furono scoperti attacchi sofisticati che sfruttavano l'assenza di timestamp o di identificatori di sessione, permettendo attacchi di tipo replay o man-in-the-middle. Questi attacchi dimostravano che l'intuizione umana, per quanto esperta, non era sufficiente a garantire la sicurezza di protocolli anche moderatamente complessi.

La motivazione principale dietro la BAN logic era quindi fornire uno strumento formale che permettesse di ragionare sistematicamente sulle credenze degli agenti che partecipano a un protocollo di autenticazione. L'idea centrale è che un protocollo è corretto se, alla fine della sua esecuzione, gli agenti coinvolti hanno le credenze corrette riguardo alle chiavi condivise, all'identità dei loro interlocutori, e alla freschezza dei messaggi scambiati. La BAN logic fornisce un linguaggio per esprimere queste credenze e un sistema di regole di inferenza per derivare le credenze che gli agenti acquisiscono man mano che il protocollo procede.

L'approccio della BAN logic è dichiaratamente simbolico e astratto. Le primitive crittografiche sono trattate come operazioni perfette: la cifratura con una chiave rende un messaggio completamente incomprensibile a chi non possiede quella chiave, e la verifica di una firma garantisce con certezza assoluta la provenienza di un messaggio. Questa astrazione, pur essendo una semplificazione della realtà, permette di concentrarsi sulla struttura logica del protocollo e sulle assunzioni implicite su cui si basa la sua sicurezza.

Il Linguaggio della BAN Logic

Il linguaggio formale della BAN logic è costruito attorno a un insieme di primitive che permettono di esprimere le credenze degli agenti riguardo a chiavi, messaggi, timestamp e identità. Il vocabolario fondamentale include i nomi degli agenti, tipicamente indicati con lettere maiuscole come A, B, S per Alice, Bob e Server, le formule che rappresentano

messaggi o proposizioni indicate con lettere greche come ϕ e ψ , e le chiavi crittografiche indicate con K o con notazioni più specifiche come K_{AB} per una chiave condivisa tra A e B .

Il cuore del linguaggio è costituito dagli operatori modali che permettono di costruire formule complesse a partire da formule semplici. L'operatore principale è $P \text{ believes } \phi$, che si legge "il principale P crede che la formula ϕ sia vera". Questo operatore cattura lo stato epistemico dell'agente P rispetto alla proposizione ϕ . È importante sottolineare che "credere" in questo contesto ha un significato tecnico preciso: non si tratta di una credenza soggettiva o probabilistica, ma di una relazione formale tra un agente e una proposizione che riflette ciò che l'agente può legittimamente dedurre dalle informazioni in suo possesso e dalle proprietà delle primitive crittografiche.

Un secondo operatore fondamentale è $P \text{ sees } X$, che significa "il principale P vede il messaggio X ", ovvero P riceve o è in possesso di X . Vedere un messaggio non implica necessariamente comprenderlo o estrarre informazione da esso: un agente può vedere un testo cifrato senza poter dedurre il messaggio in chiaro. Questo operatore cattura la distinzione cruciale tra possesso di dati e conoscenza dell'informazione in essi contenuta.

L'operatore $P \text{ said } \phi$, che si legge "il principale P ha detto ϕ ", indica che P ha inviato un messaggio che include la formula ϕ . Questo operatore non specifica quando il messaggio è stato inviato, il che richiede l'introduzione di un operatore aggiuntivo per la freschezza. Il fatto che un agente abbia detto qualcosa in passato non significa che quella affermazione sia ancora rilevante o valida nel contesto attuale del protocollo.

Per gestire la freschezza dei messaggi, la BAN logic introduce l'operatore $\text{fresh}(\phi)$, che significa "la formula ϕ è fresca", ovvero non è stata inviata in una sessione precedente del protocollo. La freschezza è cruciale per prevenire attacchi di tipo replay, dove un avversario registra messaggi di sessioni precedenti e li reinvia in un secondo momento. Un messaggio è considerato fresco se contiene un timestamp recente, un nonce generato appositamente per la sessione corrente, o se è una risposta diretta a un messaggio fresco.

Per esprimere relazioni tra agenti e chiavi, la BAN logic introduce diverse notazioni composte. L'espressione $P \text{ controls } \phi$ significa che P ha autorità su ϕ , ovvero se P crede ϕ allora anche gli altri agenti dovrebbero crederlo. Questo operatore è utilizzato tipicamente per server fidati o autorità di certificazione che hanno il potere di stabilire certi fatti, come l'associazione tra identità e chiavi pubbliche.

Le chiavi sono trattate con notazioni specifiche che indicano il tipo di chiave e gli agenti coinvolti. La notazione $P \leftrightarrow^K Q$ indica una chiave condivisa simmetrica K tra P e Q che è segreta rispetto agli altri agenti. La notazione K_P indica la chiave pubblica dell'agente P , mentre K^{-1}_P indica la corrispondente chiave privata. Una formula cifrata è scritta come $\{\phi\}_K$, che significa "il messaggio ϕ cifrato con la chiave K ".

Per esprimere la combinazione di messaggi, la BAN logic usa la notazione $\langle \phi, \psi \rangle$ per indicare la concatenazione delle formule ϕ e ψ . Questa notazione permette di rappresentare messaggi strutturati che contengono più componenti. Un'altra notazione importante è $\langle \phi \rangle_\psi$, che rappresenta il messaggio ϕ combinato con una formula ψ che serve come contesto o identificatore, utile per distinguere diversi usi dello stesso tipo di messaggio in parti diverse del protocollo.

Oltre a questi operatori principali, la BAN logic include anche alcuni operatori derivati e abbreviazioni per esprimere concetti comuni. Ad esempio, $P \text{ believes fresh}(\phi)$ e $P \text{ believes } (Q \text{ said } \phi)$ possono essere combinati per derivare $P \text{ believes } (Q \text{ believes } \phi)$, sotto certe condizioni che garantiscono che Q non avrebbe inviato ϕ se non ci credesse. Questa è la nozione di giurisdizione: se P crede che Q abbia autorità su un certo fatto, e P crede che Q abbia detto quel fatto di recente, allora P può credere al fatto stesso.

La distinzione tra chiavi pubbliche e private è gestita attraverso convenzioni notazionali e assiomi specifici. Una chiave pubblica K_P è per definizione conosciuta da tutti, mentre la chiave privata corrispondente $K^{(-1)}_P$ è conosciuta solo da P . La relazione tra queste chiavi è catturata da regole che specificano che un messaggio cifrato con una chiave privata può essere verificato da chiunque abbia la chiave pubblica corrispondente, modellando la firma digitale, mentre un messaggio cifrato con una chiave pubblica può essere decifrato solo da chi possiede la chiave privata corrispondente, modellando la cifratura a chiave pubblica.

Il linguaggio della BAN logic, pur essendo relativamente semplice rispetto ad altri formalismi logici, è sufficientemente espressivo per catturare le proprietà essenziali dei protocolli di autenticazione. La sua semplicità è stata uno dei fattori chiave del suo successo iniziale, permettendo ai progettisti di protocolli di comprendere rapidamente i concetti fondamentali e di applicare la logica alla verifica dei loro protocolli senza richiedere una formazione estensiva in logica formale.

Tuttavia, questa semplicità ha anche i suoi costi. Il linguaggio non è sufficientemente espressivo per catturare proprietà di sicurezza più sofisticate come l'anonimato, la fairness, o vincoli temporali complessi. Inoltre, la mancanza di una semantica formale precisa ha portato a diverse interpretazioni degli operatori, creando ambiguità su cosa esattamente viene dimostrato quando si verifica un protocollo con la BAN logic. Questi limiti hanno motivato lo sviluppo di estensioni e di approcci alternativi che cercano di bilanciare espressività e trattabilità.

Parte II: La BAN Logic - Analisi e Applicazioni

Le Regole di Inferenza della BAN Logic

Dopo aver definito il linguaggio formale della BAN logic, è necessario introdurre le regole di inferenza che permettono di derivare le credenze degli agenti man mano che il protocollo procede. Queste regole costituiscono il cuore del sistema deduttivo e catturano i principi fondamentali attraverso cui la conoscenza si propaga in un contesto crittografico.

La regola più fondamentale è quella della **message-meaning**, che stabilisce come un agente può interpretare un messaggio cifrato. Se un principale P crede che una chiave K sia condivisa con un altro principale Q , e P vede un messaggio cifrato con quella chiave, allora P può credere che Q abbia detto quel messaggio. Formalmente:

$P \text{ believes } (P \leftrightarrow^K Q), P \text{ sees } \{X\}_K$

P believes (Q said X)

Questa regola cattura l'intuizione che la cifratura fornisce una forma di autenticazione: solo chi possiede la chiave può aver creato quel messaggio cifrato. Nel modello simbolico della BAN logic, questa è un'inferenza valida perché si assume che la crittografia sia perfetta e che quindi solo Q, oltre a P, conosca K.

La regola del **nonce-verification** permette di derivare credenze sulla freschezza delle affermazioni. Se P crede che una formula X sia fresca, e P crede che Q abbia detto X, allora P può credere che Q crede X. La freschezza garantisce che il messaggio è stato generato nella sessione corrente del protocollo, quindi riflette le credenze attuali di Q:

P believes fresh(X), P believes (Q said X)

P believes (Q believes X)

Questa regola è cruciale per prevenire attacchi di tipo replay. Se un avversario registra un messaggio di una sessione precedente e lo reinvia, la mancanza di freschezza impedisce al ricevente di derivare credenze errate sulle intenzioni attuali del mittente.

La regola di **jurisdiction** formalizza l'autorità che certi agenti hanno su determinate affermazioni. Se P crede che Q abbia autorità su una formula X, e P crede che Q crede X, allora P può credere X:

P believes (Q controls X), P believes (Q believes X)

P believes X

Questa regola è tipicamente applicata quando si ha un'autorità fidata, come un server di autenticazione o un'autorità di certificazione, che ha il potere di stabilire certi fatti. Ad esempio, un server Kerberos ha autorità sulla distribuzione di chiavi di sessione, quindi se il server afferma che una certa chiave è condivisa tra due agenti, questi possono crederlo.

La regola della **seeing** decomposizione permette di estrarre componenti da messaggi strutturati. Se P vede una coppia di messaggi, allora P vede ciascuna componente:

P sees ⟨X, Y⟩

P sees X, P sees Y

Questa regola è semplice ma essenziale per modellare protocolli che trasmettono messaggi composti da più parti. Tuttavia, è importante notare che vedere un messaggio cifrato non implica vedere il suo contenuto: da $P \text{ sees } \{X\}_K$ non si può derivare $P \text{ sees } X$ senza conoscere la chiave K .

La regola di **freshness propagation** stabilisce che se una parte di un messaggio composto è fresca, allora l'intero messaggio è fresco:

$P \text{ believes fresh}(X)$

$P \text{ believes fresh}(\langle X, Y \rangle)$

Questa regola cattura l'idea che l'inclusione anche di una sola componente fresca, come un timestamp o un nonce, rende l'intero messaggio non riproducibile da sessioni precedenti.

Per la crittografia a chiave pubblica, la BAN logic include regole specifiche. Se P vede un messaggio cifrato con la chiave privata di Q , e P conosce la chiave pubblica di Q , allora P può dedurre che Q ha detto quel messaggio:

$P \text{ believes } K_Q^{-1} \rightarrow Q, P \text{ sees } \{X\}_{K_Q^{-1}}$

$P \text{ believes } (Q \text{ said } X)$

Questa regola modella la firma digitale: un messaggio cifrato con la chiave privata può essere verificato da chiunque con la chiave pubblica, e questa verificabilità fornisce autenticazione della provenienza.

Metodologia di Analisi dei Protocolli

L'applicazione della BAN logic alla verifica di un protocollo di autenticazione segue una metodologia sistematica che si articola in diverse fasi. Questa metodologia, pur essendo relativamente semplice da seguire, richiede attenzione ai dettagli e comprensione delle assunzioni implicite del protocollo.

La prima fase consiste nell'**idealizzazione del protocollo**. I protocolli reali contengono spesso informazioni ridondanti o dettagli implementativi che non sono rilevanti per l'analisi logica dell'autenticazione. L'idealizzazione trasforma ogni messaggio del protocollo in una formula della BAN logic che cattura il contenuto informativo essenziale, eliminando elementi come identificatori di sessione o padding che servono solo per scopi implementativi.

Ad esempio, un messaggio del tipo "A, B, N_A, {T, K_AB}K_AS" *potrebbe essere idealizzato come* " $\{T, K_{AB}\}K_{AS}$ " dove si mantiene solo la cifratura del timestamp e della chiave, eliminando i nomi degli agenti che sono implicitamente noti dal contesto. Questa fase richiede giudizio: bisogna identificare quali componenti del messaggio sono semanticamente significative per le proprietà di autenticazione che si vogliono verificare.

La seconda fase è la **specificazione delle assunzioni iniziali**. Prima che il protocollo inizi, gli agenti hanno certe credenze iniziali che costituiscono il loro stato epistemico di partenza. Queste credenze tipicamente includono: quali chiavi sono condivise con quali altri agenti, quali chiavi pubbliche corrispondono a quali agenti, quale autorità hanno certi server fidati, e quali formule sono fresche.

Ad esempio, in un protocollo che usa un server di autenticazione S, le assunzioni iniziali potrebbero includere "A believes (A $\leftrightarrow$ K_AS S)" e "A believes (S controls K_AB)" dove K_AB è una chiave di sessione che S distribuirà. Queste assunzioni devono riflettere accuratamente il modello di fiducia del protocollo: se un'assunzione è troppo forte, la verifica potrebbe avere successo anche se il protocollo è insicuro; se è troppo debole, la verifica potrebbe fallire anche per protocolli corretti.

La terza fase è l'**annotazione dei messaggi** con le formule della BAN logic. Per ogni messaggio inviato nel protocollo, si scrive la formula che rappresenta cosa l'agente ricevente vede. Questa formula usa l'operatore "sees" e include la struttura completa del messaggio, con cifrature, concatenazioni, e altre operazioni.

La quarta fase è l'**applicazione delle regole di inferenza**. Partendo dalle assunzioni iniziali e dalle formule dei messaggi visti, si applicano iterativamente le regole della BAN logic per derivare nuove credenze. Questo processo è meccanico ma richiede attenzione nel scegliere quali regole applicare e in quale ordine. L'obiettivo è derivare le credenze finali che il protocollo intende stabilire.

Per esempio, se dopo aver ricevuto un messaggio si vuole dimostrare che "A believes (A $\leftrightarrow$ K_AB B)", ovvero che A crede di condividere una chiave di sessione con B, si deve costruire una catena di inferenze che parta dalle assunzioni iniziali e dai messaggi visti, applicando regole come message-meaning, nonce-verification, e jurisdiction, fino a raggiungere la credenza desiderata.

La quinta e ultima fase è la **verifica degli obiettivi di autenticazione**. Si confrontano le credenze finali derivate con gli obiettivi di sicurezza del protocollo. Se tutte le credenze desiderate sono state derivate, il protocollo è considerato corretto rispetto a quelle proprietà di autenticazione. Se alcune credenze non possono essere derivate, questo indica una potenziale debolezza del protocollo.

È importante sottolineare che la BAN logic verifica proprietà di autenticazione ma non altre proprietà di sicurezza come la segretezza forward o la resistenza a attacchi di tipo man-in-the-middle sofisticati. La metodologia è quindi complementare ad altre forme di analisi e non sostituisce una valutazione di sicurezza completa.

Analisi del Protocollo Needham-Schroeder

Il protocollo Needham-Schroeder per la distribuzione di chiavi, proposto nel 1978, è uno dei protocolli classici più studiati nella letteratura sulla sicurezza. La sua analisi con la BAN logic illustra perfettamente sia le potenzialità che i limiti di questo approccio formale. Il protocollo originale si articola in sette messaggi tra due principali, A e B, e un server fidato S.

Il protocollo inizia con A che desidera comunicare con B. A genera un nonce N_A e lo invia al server insieme alla propria identità e quella di B:

Messaggio 1: $A \rightarrow S: A, B, N_A$

Il server S genera una chiave di sessione K_{AB} e la cifra due volte: una volta con la chiave che S condivide con A, e una volta con la chiave che S condivide con B. Il server risponde ad A con:

Messaggio 2: $S \rightarrow A: \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

Questo messaggio contiene il nonce originale (per garantire freschezza), l'identità di B, la chiave di sessione, e un "certificato" cifrato con la chiave di B che contiene la stessa chiave di sessione e l'identità di A. A può decifrare il messaggio esterno ma non il certificato interno.

A verifica che il nonce ricevuto corrisponda a quello inviato, e quindi invia a B il certificato:

Messaggio 3: $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$

B decifra il certificato e ottiene K_{AB} . Per verificare che A abbia effettivamente la chiave, B genera un nonce N_B e lo invia cifrato con K_{AB} :

Messaggio 4: $B \rightarrow A: \{N_B\}_{K_{AB}}$

A decifra il messaggio, dimostrando di possedere K_{AB} , e risponde con una funzione del nonce (tipicamente $N_B - 1$) per dimostrare che ha effettivamente processato il nonce di B:

Messaggio 5: $A \rightarrow B: \{N_B - 1\}_{K_{AB}}$

A questo punto, secondo la progettazione originale, sia A che B dovrebbero credere di condividere una chiave fresca K_{AB} .

Applichiamo ora la metodologia della BAN logic. Nella fase di idealizzazione, i messaggi vengono trasformati nelle loro rappresentazioni logiche essenziali. Il messaggio 2, ad esempio, viene idealizzato come $A \text{ sees } \{N_A, \langle K_{AB}, \{K_{AB}\}_{K_{BS}} \rangle\}_{K_{AS}}$, evidenziando la struttura della cifratura e i componenti informativamente rilevanti.

Le assunzioni iniziali includono:

- A believes ($A \leftrightarrow^K K_{AS} S$)
- B believes ($B \leftrightarrow^K K_{BS} S$)
- A believes ($S \text{ controls } (A \leftrightarrow^K K_{AB} B)$)
- B believes ($S \text{ controls } (A \leftrightarrow^K K_{AB} B)$)
- A believes fresh(N_A)

Queste assunzioni stabiliscono che A e B si fidano del server S per distribuire chiavi di sessione, e che il nonce N_A è fresco.

Analizziamo la derivazione delle credenze di A dopo il messaggio 2. A vede $\{N_A, \langle K_{AB}, \{K_{AB}\}_{K_{BS}} \rangle\}_{K_{AS}}$. Applicando la regola di message-meaning con l'assunzione che $A \leftrightarrow^A K_{AS} S$, otteniamo:

A believes (S said $\langle N_A, K_{AB} \rangle$)

Dato che A believes fresh(N_A), e il nonce appare nel messaggio, per la regola di nonce-verification:

A believes (S believes (A $\leftrightarrow^A K_{AB}$ B))

Infine, usando la regola di jurisdiction con l'assunzione che S ha autorità sulla distribuzione di chiavi:

A believes (A $\leftrightarrow^A K_{AB}$ B)

Quindi A crede correttamente di condividere K_{AB} con B. Un'analisi simile può essere condotta per B, mostrando che anche B raggiunge la credenza desiderata.

Tuttavia, la BAN logic originale non riesce a identificare un attacco sottile scoperto da Gavin Lowe nel 1995, diciassette anni dopo la pubblicazione del protocollo. L'attacco sfrutta il riuso di chiavi di sessione vecchie. Se un avversario registra il messaggio 3 di una sessione precedente dove la chiave era K_{old} , e A è stata compromessa temporaneamente permettendo all'avversario di ottenere K_{old} , l'avversario può impersonare A in una nuova sessione con B reinviando il vecchio certificato.

B, ricevendo $\{K_{old}, A\}_{K_{BS}}$, non ha modo di distinguere questo messaggio da uno fresco, perché il protocollo non include timestamp nel certificato. B crede quindi di condividere K_{old} con A, mentre A potrebbe credere di condividere una chiave completamente diversa K_{new} con B, o potrebbe non essere nemmeno consapevole della sessione.

La BAN logic non identifica questo attacco perché le sue regole di freschezza sono insufficientemente precise. La logica assume che se un nonce fresco appare in un messaggio composto, l'intero messaggio è fresco. Ma nel protocollo Needham-Schroeder, il certificato $\{K_{AB}, A\}_{K_{BS}}$ non contiene il nonce N_A , quindi non eredita la freschezza. La BAN logic standard non cattura questa distinzione: tratta il certificato come fresco per transitività, anche se in realtà può essere vecchio.

Questa limitazione non è un difetto superficiale della BAN logic, ma riflette una tensione fondamentale nel suo design. La logica è stata progettata per essere semplice e applicabile manualmente a protocolli reali, il che richiedeva un insieme limitato di regole intuitive. Rendere le regole di freschezza più precise avrebbe complicato significativamente la logica e avrebbe richiesto analisi molto più dettagliate della struttura temporale dei messaggi.

Altri Protocolli Classici

La BAN logic è stata applicata con successo a numerosi altri protocolli classici, rivelando sia la sua utilità che le sue limitazioni in contesti diversi. Un esempio significativo è il protocollo **Kerberos**, un sistema di autenticazione ampiamente utilizzato nelle reti aziendali che permette a utenti e servizi di autenticarsi reciprocamente.

Kerberos utilizza una struttura a due livelli con un Authentication Server (AS) e un Ticket Granting Server (TGS). Un client che desidera accedere a un servizio deve prima autenticarsi con l'AS, ottenere un Ticket Granting Ticket (TGT), usare questo TGT per richiedere al TGS un ticket di servizio, e infine presentare il ticket di servizio al server finale.

L'analisi con la BAN logic di Kerberos rivela alcune proprietà interessanti. Il protocollo stabilisce correttamente credenze mutue tra client e server attraverso la catena di fiducia mediata da AS e TGS. Tuttavia, la BAN logic identifica anche ridondanze: alcuni messaggi contengono informazioni che non contribuiscono alle credenze finali e potrebbero essere eliminate senza compromettere le garanzie di autenticazione.

Questa osservazione ha portato a proposte di versioni semplificate di Kerberos che mantengono le stesse proprietà di sicurezza con meno messaggi. Tuttavia, le versioni originali spesso includono queste ridondanze per ragioni pratiche non catturate dalla BAN logic, come la compatibilità con implementazioni esistenti o la gestione di casi limite in scenari di errore.

Un altro protocollo interessante è **Otway-Rees**, un protocollo di distribuzione di chiavi progettato per essere più semplice di Needham-Schroeder. L'analisi con la BAN logic mostra che Otway-Rees stabilisce correttamente credenze di autenticazione mutua e evita alcuni degli attacchi a cui Needham-Schroeder è vulnerabile. Tuttavia, il protocollo richiede più passaggi di messaggio, illustrando il trade-off tra sicurezza e efficienza.

Il protocollo **Wide-Mouthed Frog**, noto per la sua estrema semplicità (solo tre messaggi), presenta un caso interessante per la BAN logic. Il protocollo stabilisce una chiave di sessione tra due agenti attraverso un server fidato, ma fa affidamento completamente su timestamp per la freschezza, senza usare nonce generati dai partecipanti.

L'analisi con la BAN logic mostra che Wide-Mouthed Frog raggiunge i suoi obiettivi di autenticazione sotto l'assunzione che gli orologi siano sincronizzati. Tuttavia, questa assunzione è molto forte e difficile da garantire in reti distribuite, specialmente in presenza di avversari che potrebbero ritardare selettivamente messaggi. La BAN logic tratta i timestamp come nonce freschi, ma non modella esplicitamente i vincoli temporali o la possibilità di de-sincronizzazione degli orologi.

Successi e Contributi della BAN Logic

Nonostante le sue limitazioni, la BAN logic ha avuto un impatto profondo e duraturo sulla comunità della sicurezza informatica. Il suo contributo principale non è tanto nella verifica specifica di protocolli individuali, quanto nell'aver stabilito un paradigma per pensare rigorosamente all'autenticazione e nell'aver reso la verifica formale accessibile a una comunità più ampia di ricercatori e progettisti.

Prima della BAN logic, l'analisi dei protocolli di autenticazione era prevalentemente informale e basata su argomentazioni intuitive che spesso si rivelavano fallaci. La BAN logic ha introdotto un vocabolario preciso per discutere di credenze, conoscenza, e fiducia nei protocolli crittografici. Concetti come la distinzione tra "vedere" un messaggio e "credere" al suo contenuto, o l'importanza della freschezza per derivare credenze sulle intenzioni attuali degli agenti, sono diventati parte del linguaggio comune nella comunità.

La semplicità relativa della BAN logic ha permesso la sua adozione in contesti educativi. Molti corsi universitari sulla sicurezza informatica introducono studenti all'analisi formale dei protocolli attraverso la BAN logic, fornendo una prima esposizione alle idee di logica modale epistemica e di verifica formale. Questa funzione pedagogica non dovrebbe essere sottovalutata: ha formato generazioni di ricercatori che hanno poi sviluppato approcci più sofisticati.

La BAN logic ha anche stimolato la progettazione di nuovi protocolli. Diversi progettisti hanno usato i principi della BAN logic per guidare lo sviluppo di protocolli, assicurandosi che ogni messaggio contribuisse alle credenze finali desiderate e che le assunzioni di freschezza fossero adeguatamente mantenute. Alcuni protocolli sono stati esplicitamente progettati per essere "BAN-verificabili", il che ha portato a design più puliti e più facili da analizzare.

Un contributo meno ovvio ma ugualmente importante è l'aver reso evidente la necessità di formalizzazione. Gli attacchi scoperti ai protocolli classici dopo la pubblicazione della BAN logic hanno dimostrato che nemmeno i protocolli progettati da esperti e ampiamente usati erano immuni da vulnerabilità sottili. Questo ha creato consenso sulla necessità di metodi formali e ha giustificato gli investimenti in ricerca su tecniche di verifica più potenti.

La BAN logic ha anche influenzato lo sviluppo di standard industriali. Alcuni standard per protocolli di autenticazione includono ora argomentazioni semi-formali basate su concetti della BAN logic per giustificare le scelte di design. Anche quando la verifica formale completa non è eseguita, il framework concettuale della BAN logic fornisce un modo strutturato di ragionare sulla correttezza.

Infine, la BAN logic ha aperto questioni teoriche profonde sul rapporto tra logica epistemica e sicurezza, sulla formalizzazione della fiducia, e sulla semantica delle operazioni crittografiche. Queste questioni continuano a essere rilevanti nella ricerca contemporanea e hanno ispirato decenni di lavoro su logiche più espressive e su fondamenti formali della crittografia.

Parte III: Limiti della BAN Logic e Sviluppi Successivi

Le Critiche Fondamentali alla BAN Logic

Nonostante il successo iniziale e l'impatto significativo della BAN logic, la comunità scientifica ha gradualmente riconosciuto diverse limitazioni fondamentali che ne

circoscrivono l'applicabilità e l'affidabilità. Queste critiche non devono essere viste come un rifiuto del contributo della BAN logic, ma piuttosto come una maturazione della comprensione di ciò che è necessario per una verifica formale rigorosa dei protocolli di sicurezza.

La prima e più grave critica riguarda l'**assenza di una semantica formale precisa**. La BAN logic è stata presentata principalmente attraverso le sue regole sintattiche di inferenza, senza una teoria dei modelli che specifichi cosa significhi che una formula è vera in un particolare stato del sistema. Questa mancanza rende ambiguo il significato preciso degli operatori modali e lascia spazio a interpretazioni diverse.

Quando si dice che "P believes ϕ ", cosa significa esattamente? Si intende che P ha derivato ϕ attraverso inferenze logiche? Che P ha osservato evidenze che supportano ϕ ? Che P agirebbe come se ϕ fosse vero? Senza una semantica formale, queste domande non hanno risposte definitive. Diversi ricercatori hanno interpretato gli operatori in modi leggermente diversi, portando a analisi inconsistenti dello stesso protocollo.

La mancanza di semantica ha anche conseguenze pratiche per la soundness e la completeness del sistema. Un sistema logico è sound se ogni formula derivabile è vera in tutti i modelli, e complete se ogni formula vera in tutti i modelli è derivabile. Senza una nozione precisa di "modello", queste proprietà fondamentali non possono essere nemmeno formulate, tantomeno dimostrate. Questo solleva dubbi sulla correttezza matematica delle derivazioni nella BAN logic.

Una seconda critica importante riguarda il **trattamento inadeguato della freschezza**. Come abbiamo visto nell'analisi del protocollo Needham-Schroeder, le regole di freschezza della BAN logic sono troppo grossolane per catturare distinzioni temporali importanti. La logica tratta la freschezza come una proprietà binaria, fresco o non fresco, che si propaga attraverso la composizione di messaggi, ma non modella la struttura temporale precisa delle sessioni del protocollo.

In realtà, la freschezza è una nozione relativa e contestuale. Un nonce può essere fresco rispetto a una particolare sessione ma non rispetto al protocollo nel suo complesso. Un timestamp può essere considerato fresco se rientra in una certa finestra temporale, ma questa finestra potrebbe essere manipolata da un avversario che ritarda selettivamente i messaggi. La BAN logic non fornisce strumenti per ragionare su queste sfumature temporali.

Inoltre, la regola di propagazione della freschezza che afferma che se X è fresco allora $\langle X, Y \rangle$ è fresco, è semanticamente discutibile. La presenza di una componente fresca in un messaggio composto non garantisce necessariamente che tutto il messaggio sia stato generato recentemente. Y potrebbe essere un dato vecchio riutilizzato, e questo riuso potrebbe essere sfruttato in attacchi. La BAN logic non distingue tra la freschezza dell'invio del messaggio e la freschezza del suo contenuto semantico.

Una terza critica riguarda la **limitata espressività** della logica. La BAN logic può catturare proprietà di autenticazione relativamente semplici, ma non può esprimere molte proprietà di sicurezza importanti. La segretezza, ad esempio, non è direttamente esprimibile: la logica ragiona su ciò che gli agenti onesti credono, non su ciò che l'avversario può dedurre. Non esistono operatori per rappresentare esplicitamente l'avversario o le sue capacità.

La non-malleabilità, una proprietà cruciale per molti sistemi crittografici, non può essere catturata nella BAN logic. Questa proprietà richiede che un avversario che modifica un messaggio cifrato produca un testo cifrato che o non è decifrabile o contiene informazione indipendente dall'originale. Esprimere questo richiede ragionamento su tutte le possibili modifiche che l'avversario può effettuare, cosa che la BAN logic non supporta.

Proprietà di anonymity e unlinkability, sempre più rilevanti in applicazioni moderne, sono completamente fuori dalla portata della BAN logic. Queste proprietà richiedono confronti tra diverse esecuzioni del protocollo e ragionamento su equivalenze osservazionali, concetti che la BAN logic non possiede.

Una quarta critica concerne il **modello dell'avversario**. La BAN logic assume implicitamente un avversario di tipo Dolev-Yao che può intercettare, comporre e inviare messaggi, ma le capacità precise dell'avversario non sono formalizzate esplicitamente. Quali operazioni può eseguire l'avversario sui messaggi? Può generare nuove chiavi? Può corrompere agenti? Può ritardare messaggi? La logica non fornisce risposte chiare.

Questa ambiguità significa che la verifica con la BAN logic potrebbe non essere robusta rispetto a modelli di avversario leggermente diversi. Un protocollo che soddisfa i suoi obiettivi di autenticazione nella BAN logic potrebbe essere vulnerabile a un avversario con capacità non esplicitamente escluse dal modello. Senza una specifica precisa delle assunzioni sull'avversario, è difficile valutare la rilevanza pratica dei risultati di verifica.

Una quinta critica riguarda l'**idealizzazione dei protocolli**. Il passaggio dal protocollo reale alla sua rappresentazione idealizzata nella BAN logic è informale e basato sul giudizio umano. Diverse persone potrebbero idealizzare lo stesso protocollo in modi diversi, portando a conclusioni diverse sulla sua sicurezza. Non esistono linee guida formali su quali dettagli preservare e quali eliminare nell'idealizzazione.

Questa informalità è problematica perché l'idealizzazione può nascondere vulnerabilità. Se un elemento del protocollo che sembra implementativo si rivela cruciale per la sicurezza, la sua omissione nell'idealizzazione potrebbe portare a false conclusioni di correttezza. Inversamente, se l'idealizzazione introduce struttura che non è presente nel protocollo reale, potrebbe mascherare attacchi possibili nell'implementazione concreta.

Infine, c'è la questione della **composizionalità**. La BAN logic analizza protocolli in isolamento, senza considerare cosa accade quando il protocollo è composto con altri protocolli o eseguito in parallelo con se stesso in multiple sessioni. Nella pratica, i protocolli operano raramente in isolamento, e le interazioni tra sessioni o tra protocolli diversi possono creare vulnerabilità non visibili nell'analisi di singole sessioni isolate.

Un protocollo potrebbe essere sicuro quando eseguito una volta con agenti dedicati, ma insicuro quando lo stesso agente partecipa simultaneamente a multiple sessioni, potenzialmente con avversari. Questi attacchi di interleaving o di composizione parallela non sono catturati dall'analisi BAN standard, che considera implicitamente una singola esecuzione sequenziale del protocollo.

Le Estensioni della BAN Logic: GNY, AT, VO, SVO

Di fronte alle limitazioni della BAN logic, diversi ricercatori hanno proposto estensioni e varianti che cercano di affrontare specifiche critiche mantenendo lo spirito dell'approccio originale. Queste logiche estese rappresentano tentativi di bilanciare espressività, rigore formale e applicabilità pratica.

La **GNY logic**, sviluppata da Gong, Needham e Yahalom nel 1990, fu una delle prime estensioni significative. L'obiettivo principale era fornire una semantica più precisa e affrontare le ambiguità nella nozione di credenza. La GNY logic distingue esplicitamente tra diversi tipi di conoscenza: un agente può "possedere" un dato, può "riconoscere" la struttura di un messaggio, o può "credere" a un'affermazione sulla provenienza o il significato del dato.

Questa distinzione multi-livello permette analisi più sfumate. Ad esempio, un agente che riceve un messaggio cifrato possiede il testo cifrato, può riconoscere che ha una certa struttura, ad esempio è una cifratura, ma non crede necessariamente al contenuto fino a quando non lo decifra e verifica la sua provenienza. La BAN logic originale tendeva a conflare questi livelli, portando a confusione su cosa esattamente veniva dimostrato.

La GNY logic introduce anche operatori espliciti per rappresentare la convezione di un messaggio e la sua origine. Un messaggio può essere marcato come proveniente da un certo agente, e regole specifiche governano come queste marcature si propagano attraverso operazioni crittografiche. Questo permette di tracciare più precisamente la catena di custodia dei dati attraverso il protocollo.

Un contributo importante della GNY logic è l'introduzione della nozione di "recognizability". Un agente può riconoscere un messaggio se ha visto precedentemente la sua struttura o se può verificare certe proprietà senza necessariamente comprendere tutto il contenuto. Questa distinzione è particolarmente utile per analizzare protocolli dove agenti intermediari inoltrano messaggi cifrati che non possono decifrare ma possono autenticare attraverso MAC o firme digitali.

Tuttavia, la maggiore precisione della GNY logic ha un costo: la logica è significativamente più complessa della BAN logic originale, con un maggior numero di operatori e regole. Questo la rende più difficile da applicare manualmente e riduce la sua attrattiva per i progettisti di protocolli che non sono esperti di logica formale. Inoltre, molte delle ambiguità semantiche della BAN logic persistono anche nella GNY logic, solo formulate in termini diversi.

L'**AT logic**, Authentication Tests logic, proposta da Abadi e Tuttle, prende una direzione diversa. Invece di cercare di rendere più precisa la logica epistemica, la AT logic si concentra su proprietà verificabili operazionalmente. La logica definisce test espliciti che un agente può eseguire per verificare proprietà di autenticazione, e ragiona su quando questi test sono affidabili.

Un test di autenticazione potrebbe essere: "verifica che il messaggio ricevuto contiene un nonce fresco che hai generato". Se questo test ha successo, l'agente può concludere che il messaggio è stato generato dopo che il nonce è stato inviato, fornendo garanzie di freschezza. La AT logic formalizza quali conclusioni possono essere tratte da quali test in presenza di un avversario Dolev-Yao.

Questo approccio operativo ha il vantaggio di essere più concreto e più facilmente collegabile a implementazioni reali. I test corrispondono a operazioni che il codice del protocollo esegue effettivamente. Un implementatore può leggere una specifica in AT logic e tradurla direttamente in controlli nel codice, riducendo il gap tra verifica formale e implementazione.

La AT logic introduce anche una nozione esplicita di "test environment", che specifica quali informazioni sono disponibili all'agente quando esegue un test. Questo permette di ragionare su scenari dove diversi agenti hanno viste parziali e potenzialmente inconsistenti dello stato del protocollo, una situazione comune in sistemi distribuiti con comunicazione asincrona.

Tuttavia, la AT logic sacrifica parte dell'astrazione che rendeva la BAN logic attraente: invece di ragionare su credenze ad alto livello, si ragiona su operazioni di basso livello, il che può oscurare la struttura logica complessiva del protocollo. Per protocolli complessi con molti test e dipendenze intricate tra loro, l'analisi in AT logic può diventare laboriosa e difficile da seguire.

La **VO logic**, sviluppata da Van Oorschot, introduce una distinzione importante tra la source association e la recency association. La source association riguarda chi ha generato un messaggio, mentre la recency association riguarda quando è stato generato. Nella BAN logic, l'operatore "said" conflava queste due nozioni, portando a imprecisioni.

Separando queste dimensioni, la VO logic può esprimere più accuratamente proprietà come: "il messaggio è stato generato da A, source association, ma potrebbe essere stato generato in una sessione precedente, mancanza di recency association". Questo permette di identificare protocolli dove l'autenticazione della fonte è raggiunta ma la freschezza non è garantita, un problema comune in protocolli che omettono timestamp o nonce adeguati.

La VO logic introduce anche una nozione più sfumata di fiducia e autorità. Invece di un singolo operatore "controls", la logica permette di specificare diversi gradi e tipi di autorità che un agente può avere. Un server potrebbe avere autorità su alcuni tipi di chiavi ma non su altri, o potrebbe avere autorità temporanea che dipende da certe condizioni. Questa granularità permette modellazioni più realistiche di scenari complessi con multiple autorità e catene di fiducia.

Un aspetto innovativo della VO logic è il trattamento esplicito della "possession" come concetto distinto dalla "belief". Un agente può possedere una chiave senza credere che sia fresca o che sia condivisa con un particolare interlocutore. Questa distinzione è cruciale per analizzare protocolli dove chiavi possono essere compromesse o riutilizzate, e dove la semplice possessione non garantisce sicurezza.

La **SVO logic**, Syverson-Van Oorschot logic, rappresenta forse il tentativo più ambizioso di creare una logica per l'autenticazione che sia sia rigorosa che applicabile. La SVO logic parte dal riconoscimento che molte delle difficoltà con la BAN logic derivano dal tentativo di usare una logica modale per ragionare su proprietà che sono fondamentalmente operazionali e computazionali.

La SVO logic adotta un approccio ibrido. Usa logica modale per ragionare sulle credenze e sulla conoscenza degli agenti onesti, ma introduce anche regole esplicite per modellare le

capacità computazionali dell'avversario. L'avversario è rappresentato esplicitamente come un agente speciale con capacità ben definite: può intercettare messaggi su canali pubblici, può applicare funzioni a termini noti, ma non può invertire funzioni unidirezionali o decifrare senza chiavi.

Questa esplicitazione del modello dell'avversario è un passo importante verso il rigore formale. Permette di dimostrare teoremi del tipo: "se il protocollo soddisfa certe condizioni nella SVO logic, allora è sicuro contro ogni avversario con le capacità specificate". Questo collega l'analisi simbolica a nozioni di sicurezza più concrete e verificabili.

La SVO logic introduce anche una semantica più rigorosa basata su tracce. Invece di ragionare astrattamente su credenze, si considerano sequenze concrete di azioni, invii e ricezioni di messaggi, e si definisce quando una proprietà di autenticazione è soddisfatta da una traccia. Questo fornisce fondamenti matematici più solidi e permette dimostrazioni di soundness e completeness relative a questa semantica.

Un contributo tecnico importante della SVO logic è l'introduzione di "interpretation functions" che mappano formule logiche in condizioni verificabili su tracce. Questo rende esplicito il collegamento tra la sintassi della logica e il suo significato operativo, eliminando molte delle ambiguità che affliggevano la BAN logic originale.

Tuttavia, la maggiore espressività e rigore della SVO logic viene a scapito della semplicità. La logica è considerevolmente più complessa della BAN logic originale, con molte più regole e distinzioni sottili. Applicarla richiede competenza non solo in logica formale ma anche in teoria della computabilità e complessità. Questo ha limitato la sua adozione pratica, nonostante i suoi vantaggi teorici.

Un aspetto interessante comune a molte di queste estensioni è il tentativo di rendere esplicite assunzioni che nella BAN logic erano implicite o ambigue. La GNY logic esplicita i diversi livelli di conoscenza, la VO logic esplicita la distinzione tra fonte e freschezza, la SVO logic esplicita il modello dell'avversario. Questo processo di esplicitazione aumenta il rigore ma anche la complessità, illustrando il trade-off fondamentale tra semplicità e precisione che caratterizza tutta la ricerca sulle logiche per la sicurezza.

L'Emergere del Model Checking

Parallelamente allo sviluppo di logiche sempre più sofisticate per l'autenticazione, una direzione di ricerca alternativa guadagnava terreno: il model checking. Invece di derivare manualmente proprietà attraverso regole di inferenza logiche, il model checking costruisce un modello esplicito dello spazio degli stati del protocollo e verifica algebricamente se certe proprietà sono soddisfatte in tutti gli stati raggiungibili.

L'idea del model checking era stata sviluppata negli anni Ottanta per la verifica di hardware e software concorrente, dove aveva dimostrato grande successo nel trovare bug sottili in sistemi complessi. L'applicazione alla verifica dei protocolli di sicurezza sembrava naturale: un protocollo è essenzialmente un sistema concorrente dove gli agenti eseguono azioni in parallelo e comunicano attraverso messaggi.

Tuttavia, applicare il model checking ai protocolli di sicurezza presenta sfide uniche. La prima è la rappresentazione dello spazio degli stati. In un protocollo crittografico, lo stato include non solo le variabili locali degli agenti e i messaggi in transito, ma anche la conoscenza dell'avversario: tutti i termini che l'avversario ha osservato o può derivare. Questo spazio può essere infinito se l'avversario può generare arbitrariamente molti messaggi composti.

La seconda sfida è la rappresentazione dell'avversario. A differenza della verifica di software normale, dove si vuole verificare il comportamento in assenza di attacchi, nei protocolli di sicurezza si deve verificare il comportamento in presenza di un avversario attivo. L'avversario deve essere modellato come parte del sistema, con capacità ben definite di intercettare, generare e inviare messaggi.

Il breakthrough per il model checking dei protocolli di sicurezza venne dal lavoro di Dolev e Yao nel 1983, che introdusse quello che oggi è noto come il **modello Dolev-Yao** per l'avversario. In questo modello, l'avversario controlla completamente la rete: può intercettare tutti i messaggi, può generare nuovi messaggi applicando funzioni a termini che conosce, e può inviare qualsiasi messaggio generato a qualsiasi agente.

Crucialmente, il modello Dolev-Yao assume crittografia perfetta: l'avversario non può violare le primitive crittografiche. Un messaggio cifrato è completamente opaco senza la chiave corretta, e l'avversario non può estrarre informazioni da un testo cifrato oltre a riconoscere se due testi cifrati sono uguali. Questa astrazione, simile a quella della BAN logic, rende il problema di verifica più trattabile.

Il modello formalizza le capacità dell'avversario attraverso regole di deduzione. Se l'avversario conosce i termini M e N , allora conosce la coppia $\langle M, N \rangle$. Se conosce $\langle M, N \rangle$, allora conosce sia M che N . Se conosce M e la chiave K , allora conosce $\{M\}_K$. Se conosce $\{M\}_K$ e la chiave K , allora conosce M . Queste regole definiscono induttivamente l'insieme dei termini deducibili dall'avversario.

Con il modello Dolev-Yao, lo spazio degli stati del protocollo può essere rappresentato finito o semi-decidibile in molti casi interessanti. La conoscenza dell'avversario può essere caratterizzata come l'insieme dei termini deducibili dai messaggi osservati, usando regole di deduzione che riflettono le capacità dell'avversario. Questo insieme, sebbene potenzialmente infinito in termini di termini sintaticamente diversi, ha spesso una rappresentazione finita in termini di classi di equivalenza modulo le operazioni crittografiche.

Diversi strumenti di model checking per protocolli di sicurezza furono sviluppati negli anni Novanta, ciascuno con approcci diversi alla gestione dell'esplosione dello spazio degli stati. **FDR**, Failures-Divergences Refinement, basato sul process algebra CSP, rappresenta protocolli come processi che comunicano attraverso eventi sincronizzati e verifica proprietà espresse come raffinamenti di specifica.

In FDR, un protocollo è modellato come un sistema di processi CSP che comunicano attraverso canali. L'avversario è modellato come un processo speciale che può intercettare messaggi su canali pubblici e reinviarli. Una proprietà di sicurezza è espressa come una specifica di alto livello, ad esempio "il segreto non viene mai rivelato sull'interfaccia

pubblica", e si verifica se il protocollo raffina questa specifica, ovvero se ogni comportamento osservabile del protocollo è consentito dalla specifica.

FDR usa tecniche di compressione dello spazio degli stati basate su equivalenze comportamentali. Stati che hanno lo stesso comportamento osservabile esterno possono essere raggruppati, riducendo drasticamente la dimensione dello spazio da esplorare. Questo approccio è stato applicato con successo a protocolli come Needham-Schroeder e Kerberos, scoprendo vulnerabilità che erano sfuggite ad anni di analisi informale.

Murφ, un model checker esplicito, costruisce iterativamente lo spazio degli stati raggiungibili applicando tutte le possibili transizioni da ogni stato. Per gestire l'infinità potenziale dello spazio, Murφ usa tecniche di bound sulle dimensioni dei messaggi e sul numero di sessioni. Questo introduce incompletezza: attacchi che richiedono messaggi molto grandi o molte sessioni potrebbero non essere trovati.

Tuttavia, risultati teorici dimostrano che per molte classi di proprietà, se esiste un attacco, allora esiste un attacco con bound piccoli. Questo giustifica l'approccio bounded: anche se la verifica non è completa, cattura comunque gli attacchi "interessanti". Murφ è stato usato per analizzare diversi protocolli standard e ha scoperto vulnerabilità precedentemente sconosciute, incluse sottili vulnerabilità di timing e di gestione dello stato.

AVISPA, Automated Validation of Internet Security Protocols and Applications, rappresenta un approccio più integrato, combinando diverse tecniche di verifica. AVISPA permette di specificare protocolli in un linguaggio di alto livello, HLPSL, High Level Protocol Specification Language, che viene poi compilato in un formato intermedio, IF, Intermediate Format, su cui operano diversi back-end di verifica.

I back-end di AVISPA includono OFMC, on-the-fly model checker, che esplora lo spazio degli stati lazily costruendo solo gli stati necessari per verificare la proprietà, CL-AtSe, che usa tecniche di symbolic execution e constraint solving per rappresentare insiemi di stati attraverso vincoli logici, SATMC, che riduce il problema a SAT solving traducendo il protocollo e la negazione della proprietà in una formula booleana, e TA4SP, che usa tecniche di abstract interpretation per costruire approssimazioni conservative del comportamento del protocollo.

Questa architettura modulare permette di sfruttare i punti di forza di approcci diversi. Alcuni back-end sono più efficienti per certi tipi di protocolli o proprietà, e l'utente può scegliere il back-end più appropriato o eseguirli tutti per ottenere maggiore confidenza. AVISPA è stato usato per verificare numerosi protocolli industriali e ha contribuito allo sviluppo di standard di sicurezza.

Il vantaggio principale del model checking rispetto alle logiche come la BAN logic è l'automazione completa. Una volta specificato il protocollo e le proprietà da verificare, lo strumento esplora automaticamente lo spazio degli stati senza richiedere intervento umano. Questo elimina la possibilità di errori umani nell'applicazione delle regole logiche e rende la verifica accessibile anche a non-esperti di logica formale.

Inoltre, quando il model checker trova una violazione della proprietà, può produrre una traccia concreta che mostra esattamente quale sequenza di azioni porta all'attacco. Questa

traccia è immediatamente utile per capire la natura della vulnerabilità e per guidare la correzione del protocollo. Le logiche come la BAN logic, al contrario, possono solo dire che una proprietà non è derivabile, senza necessariamente indicare quale attacco è possibile.

Tuttavia, il model checking ha anche limitazioni significative. La più grave è il problema dell'esplosione dello spazio degli stati. Anche per protocolli relativamente semplici, il numero di stati raggiungibili può crescere esponenzialmente con il numero di agenti, sessioni e dimensione dei messaggi. Questo rende impossibile la verifica esaustiva di protocolli complessi o di scenari con molti partecipanti.

Per gestire questo problema, i model checker usano varie tecniche di riduzione e astrazione. Si può limitare il numero di sessioni considerate, basandosi su risultati teorici che mostrano che molte vulnerabilità sono già presenti con poche sessioni. Si può usare abstraction per raggruppare stati simili, verificando proprietà sulla rappresentazione astratta piuttosto che sugli stati concreti. Si possono sfruttare simmetrie nel protocollo per evitare di esplorare stati che sono essenzialmente equivalenti modulo ridenominazione di agenti.

Un risultato teorico importante in questo contesto è che, per molte classi di protocolli e proprietà, se esiste un attacco, allora esiste un attacco con un numero limitato di sessioni e una dimensione limitata dei messaggi. Questo giustifica l'uso di bound nei model checker: anche se la verifica è incompleta, cattura comunque tutti gli attacchi "interessanti" che non dipendono da scalare artificiosamente il numero di sessioni.

Un'altra limitazione del model checking è la dipendenza dalla specifica. La verifica è valida solo relativamente al modello specificato. Se il modello omette aspetti rilevanti del protocollo o dell'ambiente, o se le proprietà non catturano adeguatamente i requisiti di sicurezza, la verifica può dare falsi positivi, indicare sicurezza quando il protocollo è vulnerabile, o falsi negativi, segnalare attacchi che non sono realmente possibili.

Inoltre, come la BAN logic, il model checking opera tipicamente nel modello simbolico con crittografia perfetta. Le garanzie fornite riguardano la sicurezza contro avversari Dolev-Yao, non necessariamente contro avversari computazionali reali che potrebbero sfruttare debolezze nelle implementazioni delle primitive crittografiche. Questo gap tra il modello astratto e la realtà concreta è la motivazione principale per i risultati di computational soundness discussi nella Parte V.

Protocolli come Risoluzione di Equazioni

Una direzione di ricerca particolarmente fruttuosa che emerge negli anni Novanta è il riconoscimento che molti aspetti della verifica dei protocolli di sicurezza possono essere formulati come problemi di risoluzione di equazioni modulo teorie equazionali. Questa prospettiva algebrica fornisce sia nuove tecniche di verifica che insights teorici profondi sulla natura della sicurezza nei protocolli.

L'idea fondamentale è rappresentare la conoscenza dell'avversario e i messaggi del protocollo come termini in un'algebra con operazioni per cifratura, decifratura, concatenazione, proiezioni, e altre operazioni crittografiche. Le proprietà delle primitive crittografiche sono catturate da equazioni. Ad esempio, la decifratura inversa la cifratura:

$\text{dec}(\text{enc}(m, k), k) = m$. Le funzioni hash sono modellate come simboli di funzione senza equazioni, rappresentando la loro natura unidirezionale.

Un attacco al protocollo può allora essere visto come la costruzione, da parte dell'avversario, di un termine che viola una proprietà di sicurezza, usando le equazioni per manipolare termini noti. Il problema di verifica si riduce a: dati certi termini iniziali che l'avversario conosce e certe equazioni, può l'avversario derivare un termine target, ad esempio un segreto che dovrebbe rimanere privato?

Questo è un problema di deducibilità modulo teorie equazionali. La decidibilità e la complessità di questo problema dipendono criticamente dalla teoria equazionale considerata. Per teorie semplici, come la teoria vuota, nessuna equazione oltre alle uguaglianze sintattiche, o teorie con solo equazioni orientate da sinistra a destra, il problema è decidibile e algoritmi efficienti esistono.

Il problema può essere formulato più precisamente come segue. Dato un insieme finito di termini Φ che rappresenta la conoscenza iniziale dell'avversario, una teoria equazionale E che specifica le proprietà delle primitive crittografiche, e un termine target t che rappresenta il segreto, si vuole determinare se esiste una derivazione che permette di ottenere t da Φ usando le regole di deduzione dell'avversario modulo E .

Le regole di deduzione tipicamente includono: se l'avversario conosce M e N , allora conosce $\langle M, N \rangle$, se conosce $\langle M, N \rangle$ allora conosce M e N , se conosce M allora conosce $f(M)$ per ogni funzione f nella segnatura. La clausola "modulo E " significa che due termini sono considerati uguali se possono essere resi uguali applicando le equazioni di E .

Per teorie più complesse, come quelle che includono proprietà algebriche delle operazioni come associatività, commutatività, idempotenza, il problema può diventare indecidibile o avere complessità computazionale elevata. Un risultato importante di Rusinowitch e Turuani mostra che la deducibilità modulo teorie equazionali con solo equazioni convergenti, che possono essere orientate in una direzione, è decidibile, e forniscono un algoritmo basato su riscrittura di termini.

Questa prospettiva algebrica ha portato allo sviluppo di strumenti di verifica basati su tecniche di riscrittura e di risoluzione di equazioni. **Maude-NPA**, Maude-NRL Protocol Analyzer, è un esempio prominente. Maude è un linguaggio di specifica formale basato sulla riscrittura, e Maude-NPA lo estende con capacità specifiche per l'analisi dei protocolli, includendo ragionamento all'indietro per trovare attacchi.

L'analisi all'indietro parte da uno stato insicuro, ad esempio uno stato dove l'avversario ha appreso un segreto, e lavora all'indietro per determinare se esiste una sequenza di azioni che porta da uno stato iniziale legittimo allo stato insicuro. Questa tecnica può essere più efficiente della ricerca in avanti in certi casi, specialmente quando lo stato insicuro è ben caratterizzato e gli stati iniziali sono numerosi.

L'approccio all'indietro sfrutta il fatto che molte proprietà di sicurezza sono esprimibili come "uno stato cattivo non è raggiungibile". Invece di esplorare tutti i possibili stati raggiungibili in avanti dagli stati iniziali, si esplorano all'indietro i predecessori dello stato cattivo. Se nessun

predecessore è uno stato iniziale valido, allora lo stato cattivo è irraggiungibile e la proprietà è soddisfatta.

Maude-NPA usa tecniche sofisticate di narrowing, una generalizzazione dell'unificazione che combina unificazione con riscrittura, per esplorare lo spazio degli stati all'indietro modulo teorie equazionali. Questo permette di gestire proprietà algebriche complesse delle primitive crittografiche mantenendo decidibilità per classi ampie di protocolli.

Un vantaggio della prospettiva algebrica è la sua generalità. Le stesse tecniche possono essere applicate a diverse primitive crittografiche semplicemente cambiando la teoria equazionale. Non è necessario sviluppare un sistema logico completamente nuovo o modificare sostanzialmente gli algoritmi di verifica per gestire nuove primitive, purché le loro proprietà possano essere catturate equazionalmente.

Inoltre, la prospettiva algebrica collega la verifica dei protocolli a aree mature della matematica e dell'informatica teorica, come la teoria dei reticoli, l'algebra universale, e la teoria della riscrittura. Risultati e tecniche da queste aree possono essere importati e adattati alla verifica della sicurezza, arricchendo il toolkit disponibile.

Un esempio di questa fertilizzazione incrociata è l'uso di tecniche di unificazione equazionale. L'unificazione è il problema di trovare sostituzioni che rendono due termini uguali. Nell'unificazione equazionale, due termini sono considerati uguali se possono essere resi uguali applicando equazioni della teoria. Algoritmi efficienti di unificazione equazionale possono essere usati per determinare se l'avversario può costruire un termine target da termini noti.

Un risultato importante riguarda le teorie ACU, che includono equazioni per associatività, commutatività e unità di operazioni. Molte primitive crittografiche hanno queste proprietà. Ad esempio, l'operazione XOR è associativa, commutativa, e ha l'elemento zero come unità: $x \oplus 0 = x$, $x \oplus y = y \oplus x$, $(x \oplus y) \oplus z = x \oplus (y \oplus z)$. Algoritmi specializzati sono stati sviluppati per decidere l'equivalenza di termini e la deducibilità in presenza di teorie ACU, rendendo possibile la verifica automatica di protocolli che usano XOR.

L'XOR è particolarmente interessante perché introduce vulnerabilità algebriche sottili. Se un avversario osserva $\text{enc}(m_1 \oplus m_2, k)$, e conosce m_1 , può dedurre m_2 calcolando $\text{dec}(\text{enc}(m_1 \oplus m_2, k), k) \oplus m_1 = (m_1 \oplus m_2) \oplus m_1 = m_2$. Protocolli che usano XOR devono essere progettati attentamente per evitare che l'avversario possa sfruttare queste proprietà algebriche per violare la segretezza.

Tuttavia, la prospettiva algebrica ha anche limiti. Non tutte le proprietà delle primitive crittografiche possono essere catturate naturalmente da equazioni. Proprietà probabilistiche, come l'indistinguibilità computazionale, o proprietà che coinvolgono la complessità computazionale, non si prestano a rappresentazione equazionale diretta. Le equazioni catturano relazioni deterministiche tra termini, mentre molte nozioni di sicurezza crittografica sono intrinsecamente probabilistiche.

Inoltre, alcune proprietà di sicurezza sono intrinsecamente non algebriche. L'anonimato, ad esempio, richiede confrontare comportamenti di scenari diversi, non solo derivare termini. Anche se tecniche equazionali possono contribuire all'analisi, ad esempio verificando che

certi termini non sono deducibili, catturare completamente queste proprietà richiede framework più espressivi come le equivalenze osservazionali dei process calculi.

Un altro limite riguarda la complessità computazionale. Per teorie equazionali non convergenti o con proprietà algebriche complesse, il problema di deducibilità può avere complessità molto elevata o essere addirittura indecidibile. Anche quando decidibile, gli algoritmi possono essere troppo lenti per analizzare protocolli realistici. Trovare il giusto equilibrio tra espressività della teoria equazionale e trattabilità della verifica rimane una sfida aperta.

Nonostante questi limiti, la prospettiva algebrica ha contribuito significativamente alla comprensione teorica dei protocolli di sicurezza e ha ispirato lo sviluppo di strumenti pratici efficaci. La capacità di formulare problemi di sicurezza in termini algebrici precisi ha permesso di applicare tecniche matematiche potenti e di ottenere risultati di decidibilità e complessità che guidano lo sviluppo di algoritmi di verifica efficienti.

Verso Approcci Ibridi e la Sintesi delle Metodologie

L'evoluzione della ricerca sulle logiche per la sicurezza ha gradualmente portato al riconoscimento che nessun singolo approccio, che sia logica epistemica, model checking, o tecniche algebriche, è sufficiente da solo per affrontare l'intera gamma di sfide nella verifica dei protocolli di sicurezza. Questo ha stimolato lo sviluppo di approcci ibridi che combinano i punti di forza di metodologie diverse.

Tamarin è un esempio prominente di strumento ibrido che combina ragionamento simbolico con tecniche di dimostrazione automatica. Tamarin permette di specificare protocolli usando regole di riscrittura multi-set, che catturano naturalmente il comportamento concorrente di agenti multipli e la generazione di nomi freschi. Le proprietà di sicurezza sono specificate in una logica temporale di primo ordine che può esprimere sia proprietà di traccia che proprietà di equivalenza.

Tamarin usa tecniche di backward reasoning per cercare prove di sicurezza, esplorando lo spazio delle possibili derivazioni che potrebbero portare a uno stato insicuro. Quando l'esplorazione automatica non riesce a completare la prova, Tamarin permette all'utente di intervenire fornendo lemmi intermedi che guidano il prover verso la soluzione. Questo approccio interattivo bilancia automazione con controllo umano, permettendo di verificare protocolli complessi che sarebbero intrattabili per approcci completamente automatici.

Un aspetto innovativo di Tamarin è il suo supporto per teorie equazionali che modellano proprietà algebriche delle primitive crittografiche. A differenza di molti strumenti che assumono un insieme fisso di primitive, Tamarin permette all'utente di definire operazioni crittografiche arbitrarie con le loro equazioni caratteristiche. Questo ha permesso di verificare protocolli che usano primitive avanzate come cifratura Diffie-Hellman, bilinear pairings, e zero-knowledge proofs.

Un altro sviluppo importante è l'integrazione di tecniche di verifica formale con testing e analisi dinamica. Strumenti come **Scyther** combinano analisi simbolica con tecniche di bounded model checking, mentre framework come **CryptoVerif** integrano verifica simbolica

con analisi di sicurezza computazionale. Questi approcci ibridi cercano di colmare il gap tra la trattabilità del modello simbolico e il realismo del modello computazionale.

La lezione generale che emerge da questi sviluppi è che la verifica dei protocolli di sicurezza beneficia da un approccio pluralistico che usa metodologie diverse in fasi diverse del ciclo di sviluppo. Durante la progettazione iniziale, logiche semplici come la BAN logic possono fornire intuizioni rapide sui requisiti di autenticazione. Durante lo sviluppo, model checkers automatici possono scoprire vulnerabilità sottili. Per protocolli critici destinati a deployment su larga scala, tecniche più rigorose basate su dimostrazione di teoremi e computational soundness possono fornire le garanzie più forti possibili.

Questa maturità metodologica rappresenta un progresso significativo rispetto agli approcci monolitici del passato. La comunità ha sviluppato una comprensione sofisticata dei trade-off tra espressività, automazione, rigore e applicabilità, e ha creato un ricco ecosistema di strumenti e tecniche che possono essere combinate flessibilmente per affrontare problemi di verifica di diversa natura e complessità.

Prospettive Future e Conclusioni

Direzioni di Ricerca Emergenti

La ricerca sulle logiche per la sicurezza e sulla computational soundness continua a evolversi rapidamente, affrontando nuove sfide poste dall'emergere di tecnologie innovative e paradigmi crittografici che spingono i confini di ciò che è possibile computare in modo sicuro. Diverse direzioni di ricerca aperte promettono sviluppi significativi che potrebbero trasformare nuovamente il panorama della verifica formale nei prossimi anni.

La **crittografia post-quantistica** rappresenta una delle sfide più urgenti e complesse. L'avvento dei computer quantistici minaccia di rendere insicuri molti dei sistemi crittografici attualmente in uso, in particolare quelli basati sulla fattorizzazione di numeri interi (come RSA) o sul problema del logaritmo discreto (come la crittografia a curve ellittiche). In risposta, la comunità crittografica sta sviluppando primitive basate su problemi matematici che si ritiene siano resistenti anche agli attacchi quantistici, come i problemi basati su reticoli, su codici correttori di errori, su sistemi polinomiali multivariati, o su isogenie di curve ellittiche.

Queste nuove primitive hanno proprietà algebriche significativamente diverse da quelle della crittografia classica. I reticoli, ad esempio, supportano naturalmente operazioni lineari che possono essere rappresentate attraverso teorie equazionali, ma introducono anche considerazioni geometriche e di approssimazione che non hanno analoghi nella crittografia classica. Le teorie equazionali per rappresentare queste primitive nell'applied π -calculus o in altri formalismi simbolici sono ancora in fase di sviluppo.

Inoltre, molti schemi post-quantistici hanno parametri di sicurezza e prestazioni diverse da quelli classici, con messaggi più lunghi e operazioni più costose. Questo influenza la progettazione dei protocolli che li usano, introducendo nuovi trade-off tra sicurezza ed

efficienza. I framework di verifica devono essere estesi per ragionare su questi trade-off e per verificare che i protocolli mantengano sicurezza adeguata anche con i nuovi parametri.

La verifica di **protocolli blockchain e smart contract** presenta sfide uniche che combinano aspetti della verifica di protocolli distribuiti, della verifica di programmi, e della teoria dei giochi. Una blockchain è essenzialmente un registro distribuito mantenuto attraverso un protocollo di consenso che deve tollerare la presenza di partecipanti bizantini. Gli smart contract sono programmi eseguiti sulla blockchain che gestiscono asset digitali e implementano logica di business complessa.

La sicurezza di questi sistemi dipende da molteplici livelli. Il protocollo di consenso deve garantire che tutti i nodi onesti concordino sulla stessa storia delle transazioni, anche in presenza di nodi malevoli che cercano di creare biforcazioni o di censurare transazioni. Gli smart contract devono garantire proprietà come la correttezza funzionale (implementano la logica intesa), l'assenza di vulnerabilità (non possono essere sfruttati per rubare fondi o manipolare lo stato), e varie proprietà di fairness e liveness.

I framework tradizionali di verifica dei protocolli si concentrano tipicamente su protocolli con un numero fisso di partecipanti che eseguono ruoli predefiniti. Le blockchain, al contrario, hanno un insieme dinamico di partecipanti che possono unirsi e lasciare liberamente, e dove le incentivazioni economiche giocano un ruolo cruciale nel determinare il comportamento. Modellare e verificare questi sistemi richiede integrare ragionamento su protocolli crittografici con ragionamento su giochi stocastici e su comportamento strategico di agenti razionali.

Alcuni progressi sono stati fatti nell'applicare strumenti come ProVerif all'analisi di protocolli blockchain, ma questi strumenti tipicamente si concentrano su proprietà crittografiche e non catturano aspetti economici o di teoria dei giochi. Sviluppare framework che integrino verifica formale con analisi economica è una frontiera aperta importante.

L'integrazione di garanzie di **privacy differenziale** nei framework di verifica formale rappresenta un'altra direzione promettente. La privacy differenziale è diventata lo standard de facto per le garanzie quantitative di privacy, fornendo bound matematici rigorosi su quanta informazione può trapelare su individui specifici attraverso l'accesso a dati aggregati o a output di algoritmi che operano su dati sensibili.

Mentre la privacy differenziale è ben sviluppata per algoritmi statistici e database, la sua integrazione con protocolli crittografici e comunicazione sicura è meno esplorata. Come si può garantire che un protocollo di comunicazione non solo protegga la segretezza di messaggi specifici (proprietà qualitativa) ma anche limiti quantitativamente quanta informazione trapela attraverso metadati, timing di messaggi, o altre osservazioni del traffico?

I framework di verifica formale attuali sono principalmente qualitativi: un protocollo o soddisfa una proprietà di segretezza o non la soddisfa. Estenderli per ragionare su bound quantitativi sulla leakage di informazione richiede sviluppare logiche che possano esprimere e ragionare su probabilità e su misure di informazione. Alcuni lavori iniziali in questa direzione usano logiche temporali probabilistiche o logiche di informazione-flow quantitativa, ma l'integrazione con strumenti di verifica automatica rimane una sfida.

La verifica di protocolli **in presenza di side channel** e di implementazioni concrete rimane largamente irrisolta. I side channel sono canali di comunicazione non intenzionali attraverso cui informazione può trapelare da un'implementazione di un protocollo: timing di operazioni, consumo di potenza, emissioni elettromagnetiche, o anche suoni prodotti dall'hardware. Un protocollo può essere dimostrato sicuro nel modello simbolico o computazionale, ma la sua implementazione può essere vulnerabile a side channel attacks che sfruttano questi canali laterali.

Gli approcci attuali alla verifica di protocolli tipicamente assumono che le primitive crittografiche siano implementate correttamente e non trapelino informazione. Questa assunzione è spesso violata nella pratica. Ad esempio, un'implementazione di AES che usa table lookup può trapelare informazione sulla chiave attraverso cache timing attacks. Un'implementazione di RSA che usa l'algoritmo di esponenziazione square-and-multiply può trapelare la chiave attraverso timing o power analysis.

Estendere la verifica formale per considerare side channel richiede modellare non solo il comportamento funzionale del protocollo ma anche il suo comportamento temporale e le sue caratteristiche fisiche di esecuzione. Questo è estremamente complesso perché i side channel dipendono da dettagli di basso livello dell'implementazione, dell'hardware, e dell'ambiente di esecuzione che sono molto lontani dall'astrazione dei modelli simbolici o computazionali.

Alcuni progressi sono stati fatti con approcci basati su information-flow analysis che tracciano come informazione sensibile fluisce attraverso il programma e verificano che non raggiunge canali osservabili dall'avversario. Strumenti come **CacheAudit** e **ct-verif** possono verificare che implementazioni crittografiche sono constant-time, ovvero il loro tempo di esecuzione non dipende da dati segreti, prevenendo timing attacks. Tuttavia, questi strumenti operano a livello di codice o anche di assembly, molto lontano dal livello di astrazione dei protocolli, e integrarli con la verifica di protocolli di alto livello rimane una sfida aperta.

La verifica di **protocolli real-time** e con vincoli temporali precisi è un'altra area dove gli strumenti attuali sono limitati. Molti protocolli assumono implicitamente che i messaggi siano consegnati entro certi limiti di tempo, o che gli orologi dei partecipanti siano sufficientemente sincronizzati. Violare queste assunzioni può portare a vulnerabilità.

Ad esempio, protocolli che usano timestamp per freschezza possono essere vulnerabili se l'avversario può manipolare la sincronizzazione degli orologi o ritardare selettivamente i messaggi. Protocolli real-time, come quelli usati in sistemi di controllo industriale o in veicoli autonomi, hanno deadline stretti entro cui certe operazioni devono completare, e fallire questi deadline può avere conseguenze di sicurezza.

Modellare e verificare questi aspetti temporali richiede logiche temporali metriche che possano esprimere vincoli quantitativi sul tempo, e strumenti che possano ragionare su worst-case execution time e su schedulability. Alcuni lavori usano timed automata o hybrid automata per modellare protocolli real-time, ma questi approcci tendono a soffrire di esplosione dello spazio degli stati anche più severa dei model checker standard.

Integrazione di Metodi Formali nella Pratica Industriale

Oltre ai progressi teorici, un'area cruciale per l'impatto della ricerca sulle logiche per la sicurezza è l'integrazione dei metodi formali nella pratica dello sviluppo software industriale. Nonostante decenni di ricerca abbiano prodotto strumenti sofisticati e potenti, l'adozione di questi strumenti nell'industria rimane limitata. Comprendere e superare le barriere all'adozione è essenziale per realizzare il potenziale dei metodi formali.

Una barriera importante è la **curva di apprendimento**. Applicare strumenti come ProVerif o Tamarin richiede non solo competenza nell'uso dello strumento specifico, ma anche comprensione dei fondamenti teorici sottostanti: logica modale, process calculi, teorie equazionali, e modelli dell'avversario. Ingegneri software con background tradizionale raramente hanno questa formazione, e acquisirla richiede investimento significativo di tempo.

Per abbassare questa barriera, sono necessari sforzi su multipli fronti. Materiali educativi accessibili che introducono i concetti gradualmente, partendo da esempi concreti prima di introdurre formalismi astratti, possono aiutare. Interfacce user-friendly che nascondono parte della complessità dei formalismi sottostanti, permettendo agli utenti di specificare protocolli in notazioni più intuitive, possono rendere gli strumenti più accessibili. Tutorial interattivi e case study industriali possono dimostrare il valore pratico e guidare gli utenti attraverso applicazioni realistiche.

Un'altra barriera è l'**integrazione nel workflow di sviluppo**. Gli strumenti di verifica formale sono tipicamente usati in modo stand-alone, separati dagli strumenti di sviluppo, testing, e deployment che gli ingegneri usano quotidianamente. Questa separazione crea frizione: gli utenti devono esportare specifiche in formati speciali, eseguire strumenti esterni, e poi interpretare risultati in contesti diversi.

Per migliorare l'integrazione, gli strumenti di verifica dovrebbero essere incorporati nei IDE (Integrated Development Environment) che gli sviluppatori già usano, fornendo feedback in tempo reale mentre il codice viene scritto. Dovrebbero integrarsi con sistemi di continuous integration, verificando automaticamente ogni commit e segnalando problemi prima che il codice sia mergiato. Dovrebbero produrre output in formati compatibili con altri strumenti di analisi, permettendo di combinare verifica formale con testing, fuzzing, e static analysis.

La questione della **scalabilità** rimane centrale. Mentre strumenti di verifica possono gestire protocolli relativamente semplici con pochi agenti e messaggi, protocolli industriali complessi spesso coinvolgono decine di messaggi, molteplici sottoprotocolli composti, e scenari con molti partecipanti. La verifica esaustiva di questi sistemi può essere intrattabile, richiedendo giorni o settimane di computazione, o addirittura essere impossibile con le risorse disponibili.

Approcci pratici devono quindi bilanciare completezza con efficienza. Si possono verificare esaustivamente componenti critici piccoli e poi usare testing per componenti meno critici. Si possono usare astrazioni e bound per rendere la verifica trattabile, accettando che alcuni attacchi esotici potrebbero non essere scoperti. Si può concentrare la verifica su proprietà chiave invece di tentare di verificare ogni possibile proprietà.

La **gestione dei falsi positivi** è un'altra sfida pratica. Come discusso, strumenti come ProVerif usano sovra-approssimazioni che possono portare a segnalare attacchi che non sono realmente eseguibili. Quando uno strumento segnala un potenziale problema, determinare se è genuino o un artefatto dell'approssimazione richiede analisi manuale esperta, che può essere costosa e time-consuming.

Sviluppare tecniche per ridurre i falsi positivi, o per aiutare gli utenti a distinguere rapidamente falsi positivi da problemi reali, è importante per l'usabilità pratica. Visualizzazioni interattive delle tracce di attacco, con possibilità di esplorare alternative e di verificare precondizioni, possono aiutare. Raffinamenti iterativi del modello, dove l'utente può aggiungere progressivamente dettagli per eliminare tracce spurie, possono migliorare la precisione.

Infine, c'è la questione della **certificazione e compliance**. In molti settori, come finance, healthcare, o infrastrutture critiche, le organizzazioni devono dimostrare compliance con standard di sicurezza per ottenere certificazioni o per soddisfare requisiti regolatori. Se la verifica formale potesse contribuire a questi processi di certificazione, fornendo evidenze riconosciute di sicurezza, l'incentivo all'adozione aumenterebbe significativamente.

Alcune organizzazioni di standardizzazione stanno cominciando a riconoscere il valore dei metodi formali. Lo standard Common Criteria per la valutazione della sicurezza IT include livelli di assurance dove l'uso di metodi formali è raccomandato o richiesto. L'NIST (National Institute of Standards and Technology) ha pubblicato linee guida che menzionano la verifica formale come best practice per protocolli crittografici. Espandere questo riconoscimento e sviluppare framework chiari per come la verifica formale può essere usata nella certificazione è importante per promuovere l'adozione.

Riflessioni sul Percorso della Ricerca

Guardando indietro al percorso della ricerca sulle logiche per la sicurezza, dalla BAN logic pionieristica degli anni Ottanta ai sofisticati framework di computational soundness contemporanei, emergono alcune riflessioni generali sulla natura del progresso scientifico in questo campo e sulle lezioni apprese.

Una prima riflessione riguarda il **valore dell'imperfezione iniziale**. La BAN logic, con tutte le sue limitazioni e ambiguità, ha avuto un impatto enorme sulla comunità proprio perché era semplice e accessibile. Se i suoi creatori avessero atteso di sviluppare una teoria perfettamente rigorosa e completa prima di pubblicare, il campo potrebbe aver evoluto molto più lentamente. L'imperfezione iniziale ha stimolato discussione, critica costruttiva, e sforzi di miglioramento che hanno arricchito la comunità.

Questo suggerisce che in ricerca, specialmente in aree nuove, può essere più produttivo pubblicare idee promettenti anche se imperfette, invitando la comunità a contribuire al loro affinamento, piuttosto che cercare la perfezione in isolamento. Naturalmente, questo deve essere bilanciato con l'onestà intellettuale di riconoscere esplicitamente le limitazioni e di non sovrastimare le capacità degli approcci proposti.

Una seconda riflessione riguarda la **tensione tra espressività e trattabilità**. Questo è un tema ricorrente nel percorso dalle logiche semplici come la BAN logic, attraverso estensioni progressivamente più espressive come GNY e SVO, fino ai process calculi e alla computational soundness. Ogni aumento di espressività, ogni aggiunta di sfumature e precisione, ha un costo in termini di complessità, sia concettuale che computazionale.

Non esiste un punto di equilibrio ottimale universale: dipende dal contesto e dagli obiettivi. Per educazione e per comprensione concettuale di alto livello, approcci più semplici possono essere preferibili. Per analisi critica di protocolli di sicurezza nazionale, dove le conseguenze di vulnerabilità sono gravi, la massima precisione e rigore sono giustificati. Per verifica rapida durante lo sviluppo iterativo, approcci pragmatici che sacrificano completezza per velocità possono essere appropriati.

Riconoscere questa tensione e sviluppare un portfolio di approcci a diversi punti dello spettro espressività-trattabilità, invece di cercare un singolo approccio "migliore", è probabilmente la strategia più efficace. Diversi strumenti e metodologie possono coesistere e complementarsi, usati in stadi diversi del ciclo di vita del protocollo o per verificare aspetti diversi.

Una terza riflessione riguarda l'**importanza del collegamento tra teoria e pratica**. I risultati di computational soundness rappresentano un successo notevole nel costruire ponti formali tra modelli astratti e implementazioni concrete. Questi risultati dimostrano che l'astrazione non è solo convenienza euristica, ma può essere matematicamente giustificata sotto condizioni appropriate.

Tuttavia, questi collegamenti dipendono crucialmente dalle condizioni: le primitive crittografiche devono soddisfare proprietà specifiche, i protocolli devono essere usati in certi modi, l'ambiente deve rispettare certe assunzioni. Nella pratica, queste condizioni possono essere violate, intenzionalmente o accidentalmente, rompendo le garanzie teoriche.

Quindi, mentre i risultati teorici forniscono fondamenta essenziali, non possono sostituire completamente testing rigoroso, code review, e altre pratiche di ingegneria del software. Un approccio olistico alla sicurezza combina verifica formale con metodologie complementari, riconoscendo che ciascuna cattura aspetti diversi e ha punti ciechi diversi.

Una quarta riflessione riguarda il **ruolo della community e della collaborazione interdisciplinare**. I progressi più significativi nel campo sono spesso emersi dall'interazione tra ricercatori con background diversi: logici formali, crittografi teorici, esperti di verifica software, ingegneri di sistemi. Questa diversità di prospettive ha portato a insights che nessun gruppo avrebbe raggiunto isolatamente.

Ad esempio, i risultati di computational soundness richiedevano sia profonda comprensione della crittografia teorica che competenza in logica e in semantica dei linguaggi di programmazione. Lo sviluppo di strumenti pratici come ProVerif ha beneficiato di collaborazioni tra teorici che sviluppavano algoritmi e implementatori che li ottimizzavano per casi reali.

Promuovere queste collaborazioni interdisciplinari, attraverso workshop che attraggono comunità diverse, attraverso problemi di sfida che richiedono expertise multiple, e attraverso

incentivi per ricerca che attraversa confini tradizionali, è cruciale per il progresso continuo del campo.

Conclusioni

Il percorso tracciato dalle logiche specifiche per la sicurezza informatica, dalla pionieristica BAN logic attraverso estensioni sofisticate, model checking, process calculi, fino alla computational soundness, rappresenta uno dei successi più significativi dell'applicazione di metodi formali a problemi pratici di sicurezza. Questo percorso ha trasformato la verifica dei protocolli di sicurezza da un'arte oscura basata su intuizione e analisi informale in una disciplina scientifica rigorosa con fondamenta matematiche solide e strumenti pratici efficaci.

Le logiche per la sicurezza hanno fornito linguaggi precisi per ragionare su proprietà epistemiche complesse, permettendo di esprimere e verificare cosa gli agenti sanno, credono, e possono dedurre in contesti crittografici. I process calculi hanno fornito framework compositivi per modellare protocolli come sistemi concorrenti in presenza di avversari attivi. Il model checking ha reso possibile la verifica automatica di protocolli complessi, scoprendo vulnerabilità che erano sfuggite ad anni di analisi umana. I risultati di computational soundness hanno costruito ponti formali tra modelli astratti e sicurezza computazionale concreta, giustificando l'uso di astrazioni nella verifica.

Tuttavia, sfide significative rimangono aperte. L'integrazione di metodi formali nella pratica industriale è ancora limitata, con barriere culturali, educative e tecniche da superare. Nuove tecnologie come la crittografia post-quantistica, blockchain, e computazione sicura multi-party richiedono estensioni dei framework esistenti. La verifica di implementazioni concrete, considerando side channel e altri aspetti di basso livello, richiede collegare livelli di astrazione molto diversi in modi che non sono ancora pienamente compresi.

Nonostante queste sfide, il campo è vibrante e in rapida evoluzione. Nuove generazioni di ricercatori stanno affrontando problemi sempre più ambiziosi, beneficiando delle fondamenta teoriche costruite nelle decadi precedenti. Gli strumenti stanno diventando più potenti, più usabili e più ampiamente disponibili. L'integrazione di metodi formali in standard industriali e in processi di certificazione sta gradualmente aumentando.

Guardando al futuro, si può essere ottimisti che i metodi formali giocheranno un ruolo sempre più centrale nella garanzia della sicurezza dei sistemi informatici. In un mondo dove la dipendenza da sistemi digitali continua a crescere, e dove le minacce diventano sempre più sofisticate, non possiamo permetterci di affidarci solo all'intuizione e al testing per garantire la sicurezza. I metodi formali, con la loro capacità di fornire garanzie matematiche rigorose, sono essenziali per costruire sistemi veramente sicuri e affidabili.

Il trattato che si conclude qui ha cercato di tracciare il percorso storico e concettuale delle logiche per la sicurezza, dai loro inizi modesti ma rivoluzionari fino agli sviluppi contemporanei sofisticati. L'obiettivo è stato non solo di presentare i risultati tecnici, ma di illuminare le motivazioni, le sfide, le tensioni e i trade-off che hanno guidato lo sviluppo del campo. Comprendere questo contesto è essenziale per apprezzare pienamente i contributi esistenti e per essere preparati a contribuire ai prossimi capitoli di questa storia ancora in evoluzione.

La sicurezza informatica rimarrà una sfida centrale del ventunesimo secolo. Le logiche specifiche per questa sicurezza, nella loro varietà e sofisticazione, rappresentano alcuni degli strumenti più potenti che possediamo per affrontare questa sfida. Il loro sviluppo continuo, la loro integrazione con altre metodologie, e la loro adozione sempre più ampia saranno cruciali per costruire il futuro digitale sicuro che tutti desideriamo.

Parte IV: Process Calculi per la Sicurezza - Spi-Calculus e Applied Pi-Calculus

Continuazione: Fondamenti dei Process Calculi

Il π -calculus, sviluppato da Robin Milner, Joachim Parrow e David Walker nei primi anni Novanta, fornisce il substrato teorico su cui sono costruiti i process calculi per la sicurezza. La caratteristica distintiva del π -calculus rispetto ai process calculi precedenti come CCS è la mobilità dei nomi: i nomi dei canali di comunicazione possono essere trasmessi come dati attraverso i canali stessi, permettendo di modellare topologie di comunicazione dinamiche dove nuovi canali vengono creati e le connessioni tra processi evolvono durante l'esecuzione.

Nel π -calculus, un processo è costruito componendo processi atomici usando operatori di composizione. I processi atomici fondamentali sono: il processo nullo 0 che non fa nulla e rappresenta la terminazione, l'output di un valore su un canale scritto come $c\langle v \rangle.P$ che invia il valore v sul canale c e poi continua come il processo P , e l'input da un canale scritto come $c(x).P$ che riceve un valore sul canale c , lo lega alla variabile x , e poi continua come P con x sostituito dal valore ricevuto.

Gli operatori di composizione includono: la composizione parallela $P \mid Q$ che rappresenta i processi P e Q in esecuzione concorrente, la replicazione $!P$ che rappresenta un numero illimitato di copie del processo P in esecuzione parallela, la restrizione $(\nu n)P$ che crea un nuovo nome n con scope limitato al processo P , e il processo condizionale $\text{if } \phi \text{ then } P \text{ else } Q$ che esegue P se la condizione ϕ è vera, altrimenti esegue Q .

La semantica operativa del π -calculus è definita attraverso una relazione di transizione che specifica come i processi evolvono. La regola fondamentale è la comunicazione: se un processo offre un output su un canale e un altro processo offre un input sullo stesso canale, i due possono sincronizzarsi, con il valore trasmesso dall'output che viene ricevuto dall'input. Formalmente:

$$c\langle v \rangle.P \mid c(x).Q \rightarrow P \mid Q\{v/x\}$$

dove $Q\{v/x\}$ denota il processo Q con tutte le occorrenze libere di x sostituite da v .

Le altre regole della semantica operativa governano come le transizioni si propagano attraverso gli operatori di composizione. La composizione parallela permette a ciascun componente di fare transizioni indipendentemente, la restrizione preserva lo scope dei nomi, la replicazione può sempre generare nuove copie del processo replicato, e il condizionale valuta la condizione e sceglie il ramo appropriato.

Una caratteristica importante del π -calculus è la nozione di equivalenza osservazionale, che cattura l'idea che due processi sono equivalenti se non possono essere distinti da un osservatore esterno che interagisce con loro. Due processi P e Q sono osservazionalmente equivalenti se in ogni contesto, ovvero messi in parallelo con qualsiasi altro processo, si comportano allo stesso modo rispetto alle interazioni osservabili. Questa nozione di equivalenza è fondamentale per definire proprietà di sicurezza come la segretezza e l'anonimato in termini comportamentali.

Il Spi-Calculus: Estendere il Pi-Calculus con Crittografia

Il spi-calculus, dove "spi" sta per "security protocols in pi-calculus", fu introdotto da Martin Abadi e Andrew Gordon nel 1997 come estensione del π -calculus specificamente progettata per modellare protocolli crittografici. L'idea centrale è aggiungere al π -calculus costrutti primitivi per la cifratura e la decifratura simmetrica, insieme a regole che specificano come questi costrutti interagiscono con il resto del calcolo.

La sintassi del spi-calculus estende quella del π -calculus con termini per rappresentare messaggi cifrati e processi per cifrare e decifrare. Un messaggio cifrato è scritto come $\{M\}_K$ dove M è il messaggio in chiaro e K è la chiave. I processi di cifratura e decifratura sono espressi attraverso pattern matching: un processo può cercare di decifrare un messaggio ricevuto specificando il pattern che si aspetta.

La forma più semplice di decifratura è scritta come:

case L of $\{x\}_K$ in P

che cerca di decifrare L con la chiave K : se L ha la forma $\{M\}_K$ per qualche messaggio M , allora il processo continua come P con x sostituito da M ; altrimenti il processo si blocca, indicando un fallimento nella decifratura.

Il spi-calculus include anche pattern più complessi per gestire messaggi strutturati. Ad esempio:

case L of $\{x, y\}_K$ in P

cerca di decifrare L e verificare che il contenuto sia una coppia, legando le due componenti a x e y rispettivamente.

La semantica del spi-calculus estende quella del π -calculus con regole per la cifratura e decifratura. La regola fondamentale è che un messaggio $\{M\}_K$ può essere decifrato solo se si possiede la chiave K . Formalmente, un processo che esegue case $\{M\}_K$ of $\{x\}_K$ in P può fare una transizione verso $P\{M/x\}$, mentre un processo che esegue case $\{M\}_K$ of $\{x\}_{K'}$ in P dove $K \neq K'$ non può fare alcuna transizione e si blocca.

Questa semantica cattura l'assunzione di crittografia perfetta: un messaggio cifrato è completamente opaco senza la chiave corretta. Nel modello simbolico, non esiste la possibilità di attacchi crittanalitici, di cifrature parzialmente deboli, o di informazioni che trapelano dalla dimensione del messaggio o da altri canali laterali. Questa astrazione

permette di concentrarsi sulla struttura del protocollo indipendentemente dai dettagli implementativi delle primitive crittografiche.

Una caratteristica importante del spi-calculus è come tratta la conoscenza dell'avversario. Nel framework del π -calculus, l'avversario è modellato come un processo che può interagire con i processi onesti attraverso canali pubblici. L'avversario può ascoltare tutti i messaggi inviati su canali pubblici, può generare nuovi messaggi usando i termini che ha appreso, e può inviare questi messaggi ai processi onesti. Tuttavia, l'avversario non può violare le primitive crittografiche: non può decifrare un messaggio senza la chiave, non può estrarre componenti da un messaggio cifrato, e non può generare messaggi cifrati con chiavi che non possiede.

La proprietà di segretezza nel spi-calculus è definita usando l'equivalenza osservazionale. Un termine M è segreto rispetto a un avversario A se il processo che esegue il protocollo e poi rivela M è osservazionalmente equivalente al processo che esegue il protocollo e poi rivela un termine diverso M' . Intuitivamente, se l'avversario non può distinguere tra questi due scenari, allora non ha appreso nulla su M attraverso l'esecuzione del protocollo.

Questa definizione di segretezza basata sull'equivalenza è molto potente e cattura nozioni sofisticate. Ad esempio, cattura automaticamente la segretezza forward: se un protocollo stabilisce una chiave di sessione che rimane segreta anche quando chiavi a lungo termine vengono successivamente compromesse, questo si riflette nel fatto che l'avversario non può distinguere la chiave di sessione reale da una chiave casuale anche avendo accesso alle chiavi a lungo termine.

Abadi e Gordon svilupparono anche tecniche per verificare la segretezza nel spi-calculus basate su sistemi di tipi. L'idea è assegnare tipi ai canali e ai messaggi in modo che il sistema di tipi garantisca che certi dati non possano fluire verso certi destinatari. Se un protocollo è ben tipato secondo questo sistema, allora certe proprietà di segretezza sono automaticamente garantite. Questo approccio basato sui tipi ha il vantaggio di essere compositivo: se singoli componenti di un sistema sono ben tipati, allora anche la loro composizione è ben tipata, e le proprietà di sicurezza si compongono automaticamente.

Il sistema di tipi per il spi-calculus usa una nozione di tipi di canale che specifica quali tipi di valori possono essere trasmessi su un canale e con quale livello di sicurezza. Un canale può essere dichiarato "pubblico", accessibile all'avversario, o "privato", accessibile solo a certi processi onesti. Un valore può essere dichiarato "alto", sensibile, o "basso", pubblico. Il sistema di tipi garantisce che valori alti non fluiscano su canali pubblici, fornendo una garanzia di non-interferenza: l'esecuzione del protocollo su canali pubblici non dipende da valori alti, quindi l'avversario non può imparare nulla su questi valori.

Tuttavia, il sistema di tipi del spi-calculus ha limitazioni. È ristretto alla cifratura simmetrica e non gestisce direttamente la crittografia a chiave pubblica, le firme digitali, o altre primitive crittografiche avanzate. Inoltre, può verificare solo certe proprietà di segretezza e ha difficoltà con proprietà più complesse come l'autenticazione o la non-malleabilità. Queste limitazioni motivarono lo sviluppo dell'applied π -calculus, una generalizzazione più flessibile.

L'Applied Pi-Calculus: Un Framework Generale

L'applied π -calculus, introdotto da Martín Abadi e Cédric Fournet nel 2001, generalizza il spi-calculus permettendo di definire teorie equazionali arbitrarie per rappresentare le primitive crittografiche e le loro proprietà algebriche. Invece di avere costrutti specifici per la cifratura e decifratura hard-coded nella sintassi del calcolo, l'applied π -calculus permette di specificare una segnatura di simboli di funzione, rappresentanti operazioni come cifratura, decifratura, firma, verifica, hash, insieme a un insieme di equazioni che specificano le relazioni algebriche tra queste funzioni.

La sintassi dell'applied π -calculus è simile a quella del π -calculus standard, ma arricchita con la capacità di applicare funzioni a termini. Un termine può essere un nome, che rappresenta un canale, una chiave, un nonce, una variabile, o un'applicazione di funzione $f(M_1, \dots, M_n)$ dove f è un simbolo di funzione della segnatura e $M_1, \dots, M_n$ sono termini.

Le equazioni nella teoria equazionale specificano quali termini dovrebbero essere considerati uguali. Ad esempio, per modellare la cifratura simmetrica, si potrebbero definire due funzioni enc e dec con l'equazione:

$$\text{dec}(\text{enc}(m, k), k) = m$$

che specifica che decifrare un testo cifrato con la chiave corretta restituisce il messaggio originale. Per modellare la crittografia a chiave pubblica, si potrebbero definire funzioni pk per derivare la chiave pubblica dalla chiave privata, aenc per cifratura asimmetrica, e adec per decifratura asimmetrica, con equazioni:

$$\text{adec}(\text{aenc}(m, \text{pk}(k)), k) = m$$

Per modellare le firme digitali, si potrebbero definire funzioni sign e verify con equazione:

$$\text{verify}(\text{sign}(m, k), m, \text{pk}(k)) = \text{true}$$

Questa flessibilità permette di modellare un'ampia varietà di primitive crittografiche usando lo stesso framework. Si possono modellare funzioni hash come simboli di funzione senza equazioni, rappresentando funzioni unidirezionali, commitment schemes con equazioni che permettono di aprire un commitment, schemi di cifratura omomorfica con equazioni che permettono operazioni sui testi cifrati, e molte altre costruzioni.

Un aspetto importante dell'applied π -calculus è come gestisce la valutazione dei termini. Quando un processo calcola un termine, il calcolo applica le equazioni della teoria equazionale per ridurre il termine a una forma normale. Ad esempio, se un processo calcola $\text{dec}(\text{enc}(m, k), k)$, questo termine viene automaticamente semplificato a m usando l'equazione appropriata. Tuttavia, se un processo calcola $\text{dec}(\text{enc}(m, k), k')$ dove $k \neq k'$, il termine non può essere ulteriormente ridotto e rimane come un termine bloccato che rappresenta un fallimento della decifratura.

La semantica dell'applied π -calculus è definita in modo che processi che differiscono solo per termini equivalenti secondo la teoria equazionale siano considerati equivalenti. Questo permette di ragionare su protocolli al giusto livello di astrazione, ignorando dettagli implementativi irrilevanti che sono catturati dalle equazioni.

L'applied π -calculus introduce anche il concetto di frame, che rappresenta l'insieme di messaggi che un avversario ha osservato durante l'esecuzione del protocollo. Un frame è essenzialmente una sostituzione che associa variabili, rappresentanti canali osservabili dall'avversario, ai messaggi trasmessi su quei canali. L'analisi della sicurezza si riduce quindi a ragionare su quali termini l'avversario può costruire dato un certo frame.

La capacità dell'avversario di costruire termini è formalizzata attraverso il concetto di deducibilità statica. Dato un frame ϕ e un termine M , si chiede: può l'avversario costruire M usando i termini nel frame ϕ e applicando funzioni della segnatura? Formalmente, questo è espresso come $\phi \vdash M$, il frame ϕ deduce M . Le regole di deduzione permettono all'avversario di applicare funzioni a termini noti, di usare nomi pubblici, e di usare termini che appaiono esplicitamente nel frame.

Due proprietà di sicurezza fondamentali possono essere definite in termini di deducibilità. La segretezza di un termine segreto s rispetto a un frame ϕ è la condizione che $\phi \not\vdash s$, ovvero l'avversario non può dedurre s dai messaggi osservati. L'autenticità di un canale c rispetto a un protocollo P è la condizione che se l'avversario invia un messaggio su c che viene accettato da un processo onesto, allora quel messaggio deve essere stato originato da un processo onesto in quella sessione del protocollo.

Oltre alle proprietà basate sulla deducibilità, l'applied π -calculus permette di definire proprietà di sicurezza in termini di equivalenza osservazionale o di equivalenze più deboli come la trace equivalence. Due processi P e Q sono trace equivalent se, per ogni sequenza di azioni osservabili che P può eseguire producendo un frame ϕ , esiste una sequenza di azioni che Q può eseguire producendo un frame ψ tale che ϕ e ψ sono indistinguibili dall'avversario, ovvero per ogni termine M , $\phi \vdash M$ se e solo se $\psi \vdash M$.

Questa nozione di equivalenza è molto potente e permette di esprimere proprietà sofisticate. L'anonimato può essere espresso dicendo che un processo dove Alice esegue un'azione è trace equivalent a un processo dove Bob esegue la stessa azione: se l'avversario non può distinguere i due scenari, allora non può determinare chi ha eseguito l'azione. La non-malleabilità può essere espressa dicendo che un avversario che modifica un messaggio cifrato produce un testo cifrato che o non è decifrabile o contiene un messaggio indipendente dall'originale.

L'applied π -calculus ha ispirato lo sviluppo di numerosi strumenti di verifica automatica. Il più prominente è ProVerif, sviluppato da Bruno Blanchet, che implementa un algoritmo di verifica basato su abstract interpretation e risoluzione. ProVerif può verificare automaticamente sia proprietà di traccia, come la segretezza e certe forme di autenticazione, che proprietà di equivalenza, come anonimato e strong secrecy, anche per protocolli con un numero illimitato di sessioni.

L'algoritmo di ProVerif funziona traducendo il protocollo specificato nell'applied π -calculus in un insieme di clausole di Horn che rappresentano sovra-approssimazioni delle capacità dell'avversario e dei possibili comportamenti del protocollo. La verifica di una proprietà di segretezza si riduce quindi a determinare se una certa clausola, corrispondente all'avversario che apprende il segreto, è derivabile dall'insieme di clausole. Questa verifica può essere effettuata automaticamente usando tecniche di risoluzione, e ProVerif

implementa ottimizzazioni sofisticate per gestire efficientemente clausole complesse che emergono da protocolli realistici.

Per proprietà di equivalenza, ProVerif usa un approccio basato su bi-processi: invece di verificare separatamente che P e Q sono equivalenti, si costruisce un bi-processo che esegue simultaneamente P e Q , scegliendo non deterministicamente quale processo simulare in ogni punto della computazione. Se il bi-processo preserva certe invarianti che garantiscono che l'avversario vede la stessa sequenza di messaggi indipendentemente da quale ramo è stato scelto, allora P e Q sono equivalenti.

L'approccio di ProVerif ha alcuni limiti. Poiché usa sovra-approssimazioni, può produrre falsi allarmi: segnalare possibili attacchi che in realtà non sono eseguibili nel protocollo reale. Quando ProVerif riporta un potenziale attacco, è necessario esaminare manualmente la traccia per determinare se rappresenta un attacco genuino o un artefatto della sovra-approssimazione. Inoltre, ProVerif non gestisce alcune proprietà che richiedono ragionamenti complessi su equazioni, come certe forme di fairness o proprietà che dipendono da vincoli numerici o temporali precisi.

Nonostante questi limiti, ProVerif è stato applicato con successo a numerosi protocolli reali, inclusi protocolli standardizzati come TLS, SSH, Kerberos, e protocolli più avanzati come quelli usati in sistemi di voto elettronico e in applicazioni blockchain. Ha scoperto vulnerabilità precedentemente sconosciute in diverse implementazioni e ha contribuito significativamente alla fiducia nella sicurezza di protocolli ampiamente utilizzati.

Teorie Equazionali e Proprietà Algebriche

Un aspetto cruciale dell'applied π -calculus è la scelta delle teorie equazionali per rappresentare le primitive crittografiche. Equazioni diverse catturano proprietà algebriche diverse, e la scelta delle equazioni può influenzare significativamente sia le proprietà di sicurezza verificabili che la complessità della verifica.

Le teorie equazionali più semplici sono quelle che includono solo equazioni che permettono di disfare operazioni crittografiche in condizioni appropriate, come l'equazione $\text{dec}(\text{enc}(m, k), k) = m$ per la decifratura. Queste teorie, chiamate teorie convergenti o canoniche, hanno la proprietà che ogni termine può essere ridotto a una forma normale unica applicando le equazioni in una direzione, da sinistra a destra. Questa proprietà rende la verifica più trattabile poiché permette di decidere efficientemente se due termini sono equivalenti.

Tuttavia, molte primitive crittografiche hanno proprietà algebriche più complesse che non possono essere catturate da teorie convergenti. Ad esempio, la cifratura omomorfica ha la proprietà che $\text{enc}(m_1, k) \oplus \text{enc}(m_2, k) = \text{enc}(m_1 \oplus m_2, k)$, dove $\oplus$ rappresenta un'operazione, come l'addizione o la moltiplicazione, che può essere eseguita sia sui messaggi in chiaro che sui testi cifrati. Questa proprietà è naturalmente espressa da un'equazione, ma l'equazione non è orientabile in una direzione preferita: non è chiaro se dovrebbe essere applicata da sinistra a destra o da destra a sinistra.

Equazioni non orientabili portano a teorie equazionali più complesse dove possono esistere multiple forme normali per lo stesso termine, o dove la riduzione a forma normale può richiedere di applicare equazioni in entrambe le direzioni. Questo rende la verifica

significativamente più difficile, poiché decidere se due termini sono equivalenti può richiedere di esplorare uno spazio di possibili riduzioni, e in generale il problema di equivalenza di termini può diventare indecidibile per teorie equazionali arbitrarie.

Ricercatori hanno sviluppato tecniche per gestire classi specifiche di teorie equazionali non convergenti che sono comunque trattabili. Le teorie ACU, che includono equazioni per associatività, commutatività e unità di operazioni, sono particolarmente importanti perché molte primitive crittografiche hanno queste proprietà. Ad esempio, l'operazione XOR è associativa, commutativa, e ha l'elemento zero come unità. Algoritmi specializzati sono stati sviluppati per decidere l'equivalenza di termini e la deducibilità in presenza di teorie ACU.

Un'altra classe importante di teorie sono quelle che modellano funzioni distruttive, dove l'applicazione di certe funzioni può annullare gli effetti di altre funzioni non solo direttamente ma attraverso catene di operazioni. Ad esempio, in presenza di funzioni per la cifratura omomorfa, un avversario potrebbe essere in grado di derivare la differenza tra due messaggi cifrati anche senza conoscere i messaggi in chiaro, semplicemente sottraendo i testi cifrati. Gestire correttamente queste interazioni richiede algoritmi sofisticati che esplorino tutte le possibili derivazioni che l'avversario può effettuare.

La scelta della teoria equazionale influenza anche quali attacchi possono essere scoperti. Una teoria troppo astratta potrebbe mascherare vulnerabilità che dipendono da proprietà algebriche specifiche delle primitive. Una teoria troppo dettagliata potrebbe rendere la verifica intrattabile. Trovare il giusto equilibrio richiede comprensione sia delle proprietà matematiche delle primitive che degli obiettivi di sicurezza del protocollo. del protocollo su canali pubblici non dipende da valori alti, quindi l'avversario non può imparare nulla su questi valori.

Tuttavia, il sistema di tipi del spi-calculus ha limitazioni. È ristretto alla cifratura simmetrica e non gestisce direttamente la crittografia a chiave pubblica, le firme digitali, o altre primitive crittografiche avanzate. Inoltre, può verificare solo certe proprietà di segretezza e ha difficoltà con proprietà più complesse come l'autenticazione o la non-malleabilità. Queste limitazioni motivarono lo sviluppo dell'applied π -calculus, una generalizzazione più flessibile.

L'Applied Pi-Calculus: Un Framework Generale

L'applied π -calculus, introdotto da Martín Abadi e Cédric Fournet nel 2001, generalizza il spi-calculus permettendo di definire teorie equazionali arbitrarie per rappresentare le primitive crittografiche e le loro proprietà algebriche. Invece di avere costrutti specifici per la cifratura e decifratura hard-coded nella sintassi del calcolo, l'applied π -calculus permette di specificare una segnatura di simboli di funzione (rappresentanti operazioni come cifratura, decifratura, firma, verifica, hash, ecc.) insieme a un insieme di equazioni che specificano le relazioni algebriche tra queste funzioni.

La sintassi dell'applied π -calculus è simile a quella del π -calculus standard, ma arricchita con la capacità di applicare funzioni a termini. Un termine può essere un nome (che rappresenta un canale, una chiave, un nonce, ecc.), una variabile, o un'applicazione di funzione $f(M_1, \dots, M_n)$ dove f è un simbolo di funzione della segnatura e $M_1, \dots, M_n$ sono termini.

Le equazioni nella teoria equazionale specificano quali termini dovrebbero essere considerati uguali. Ad esempio, per modellare la cifratura simmetrica, si potrebbero definire due funzioni enc e dec con l'equazione:

$$\text{dec}(\text{enc}(m, k), k) = m$$

che specifica che decifrare un testo cifrato con la chiave corretta restituisce il messaggio originale. Per modellare la crittografia a chiave pubblica, si potrebbero definire funzioni pk (per derivare la chiave pubblica dalla chiave privata), aenc (cifratura asimmetrica), e adec (decifratura asimmetrica), con equazioni:

$$\text{adec}(\text{aenc}(m, \text{pk}(k)), k) = m$$

Per modellare le firme digitali, si potrebbero definire funzioni sign e verify con equazione:

$$\text{verify}(\text{sign}(m, k), m, \text{pk}(k)) = \text{true}$$

Questa flessibilità permette di modellare un'ampia varietà di primitive crittografiche usando lo stesso framework. Si possono modellare funzioni hash (come simboli di funzione senza equazioni, rappresentando funzioni unidirezionali), commitment schemes (con equazioni che permettono di aprire un commitment), schemi di cifratura omomorfica (con equazioni che permettono operazioni sui testi cifrati), e molte altre costruzioni.

Un aspetto importante dell'applied π -calculus è come gestisce la valutazione dei termini. Quando un processo calcola un termine, il calcolo applica le equazioni della teoria equazionale per ridurre il termine a una forma normale. Ad esempio, se un processo calcola $\text{dec}(\text{enc}(m, k), k)$, questo termine viene automaticamente semplificato a m usando l'equazione appropriata. Tuttavia, se un processo calcola $\text{dec}(\text{enc}(m, k), k')$ dove $k \neq k'$, il termine non può essere ulteriormente ridotto e rimane come un termine "bloccato" che rappresenta un fallimento della decifratura.

La semantica dell'applied π -calculus è definita in modo che processi che differiscono solo per termini equivalenti secondo la teoria equazionale siano considerati equivalenti. Questo permette di ragionare su protocolli al giusto livello di astrazione, ignorando dettagli implementativi irrilevanti che sono catturati dalle equazioni.

L'applied π -calculus introduce anche il concetto di frame, che rappresenta l'insieme di messaggi che un avversario ha osservato durante l'esecuzione del protocollo. Un frame è essenzialmente una sostituzione che associa variabili (rappresentanti canali osservabili dall'avversario) ai messaggi trasmessi su quei canali. L'analisi della sicurezza si riduce quindi a ragionare su quali termini l'avversario può costruire dato un certo frame.

La capacità dell'avversario di costruire termini è formalizzata attraverso il concetto di deducibilità statica. Dato un frame ϕ e un termine M , si chiede: può l'avversario costruire M usando i termini nel frame ϕ e applicando funzioni della segnatura? Formalmente, questo è espresso come $\phi \vdash M$ (il frame ϕ deduce M). Le regole di deduzione permettono all'avversario di applicare funzioni a termini noti, di usare nomi pubblici, e di usare termini che appaiono esplicitamente nel frame.

Due proprietà di sicurezza fondamentali possono essere definite in termini di deducibilità. La segretezza di un termine segreto s rispetto a un frame ϕ è la condizione che $\phi \not\vdash s$, ovvero l'avversario non può dedurre s dai messaggi osservati. L'autenticità di un canale c rispetto a un protocollo P è la condizione che se l'avversario invia un messaggio su c che viene accettato da un processo onesto, allora quel messaggio deve essere stato originato da un processo onesto in quella sessione del protocollo.

Oltre alle proprietà basate sulla deducibilità, l'applied π -calculus permette di definire proprietà di sicurezza in termini di equivalenza osservazionale o di equivalenze più deboli come la trace equivalence. Due processi P e Q sono trace equivalent se, per ogni sequenza di azioni osservabili che P può eseguire producendo un frame ϕ , esiste una sequenza di azioni che Q può eseguire producendo un frame ψ tale che ϕ e ψ sono indistinguibili dall'avversario (ovvero, per ogni termine M , $\phi \vdash M$ se e solo se $\psi \vdash M$).

Questa nozione di equivalenza è molto potente e permette di esprimere proprietà sofisticate. L'anonimato può essere espresso dicendo che un processo dove Alice esegue un'azione è trace equivalent a un processo dove Bob esegue la stessa azione: se l'avversario non può distinguere i due scenari, allora non può determinare chi ha eseguito l'azione. La non-malleabilità può essere espressa dicendo che un avversario che modifica un messaggio cifrato produce un testo cifrato che o non è decifrabile o contiene un messaggio indipendente dall'originale.

L'applied π -calculus ha ispirato lo sviluppo di numerosi strumenti di verifica automatica. Il più prominente è ProVerif, sviluppato da Bruno Blanchet, che implementa un algoritmo di verifica basato su abstract interpretation e risoluzione. ProVerif può verificare automaticamente sia proprietà di traccia (come la segretezza e certe forme di autenticazione) che proprietà di equivalenza (come anonimato e strong secrecy), anche per protocolli con un numero illimitato di sessioni.

L'algoritmo di ProVerif funziona traducendo il protocollo specificato nell'applied π -calculus in un insieme di clausole di Horn che rappresentano sovra-approssimazioni delle capacità dell'avversario e dei possibili comportamenti del protocollo. La verifica di una proprietà di segretezza si riduce quindi a determinare se una certa clausola (corrispondente all'avversario che apprende il segreto) è derivabile dall'insieme di clausole. Questa verifica può essere effettuata automaticamente usando tecniche di risoluzione, e ProVerif implementa ottimizzazioni sofisticate per gestire efficientemente clausole complesse che emergono da protocolli realistici.

Per proprietà di equivalenza, ProVerif usa un approccio basato su bi-processi: invece di verificare separatamente che P e Q sono equivalenti, si costruisce un bi-processo che esegue "simultaneamente" P e Q , scegliendo non deterministicamente quale processo simulare in ogni punto della computazione. Se il bi-processo preserva certe invarianti che garantiscono che l'avversario vede la stessa sequenza di messaggi indipendentemente da quale ramo è stato scelto, allora P e Q sono equivalenti.

L'approccio di ProVerif ha alcuni limiti. Poiché usa sovra-approssimazioni, può produrre falsi allarmi: segnalare possibili attacchi che in realtà non sono eseguibili nel protocollo reale. Quando ProVerif riporta un potenziale attacco, è necessario esaminare manualmente la traccia per determinare se rappresenta un attacco genuino o un artefatto della

sovra-approssimazione. Inoltre, ProVerif non gestisce alcune proprietà che richiedono ragionamenti complessi su equazioni, come certe forme di fairness o proprietà che dipendono da vincoli numerici o temporali precisi.

Nonostante questi limiti, ProVerif è stato applicato con successo a numerosi protocolli reali, inclusi protocolli standardizzati come TLS, SSH, Kerberos, e protocolli più avanzati come quelli usati in sistemi di voto elettronico e in applicazioni blockchain. Ha scoperto vulnerabilità precedentemente sconosciute in diverse implementazioni e ha contribuito significativamente alla fiducia nella sicurezza di protocolli ampiamente utilizzati.

Teorie Equazionali e Proprietà Algebriche

Un aspetto cruciale dell'applied π -calculus è la scelta delle teorie equazionali per rappresentare le primitive crittografiche. Equazioni diverse catturano proprietà algebriche diverse, e la scelta delle equazioni può influenzare significativamente sia le proprietà di sicurezza verificabili che la complessità della verifica.

Le teorie equazionali più semplici sono quelle che includono solo equazioni che permettono di "disfare" operazioni crittografiche in condizioni appropriate, come l'equazione $\text{dec}(\text{enc}(m, k), k) = m$ per la decifratura. Queste teorie, chiamate teorie convergenti o canoniche, hanno la proprietà che ogni termine può essere ridotto a una forma normale unica applicando le equazioni in una direzione (da sinistra a destra). Questa proprietà rende la verifica più trattabile poiché permette di decidere efficientemente se due termini sono equivalenti.

Tuttavia, molte primitive crittografiche hanno proprietà algebriche più complesse che non possono essere catturate da teorie convergenti. Ad esempio, la cifratura omomorfica ha la proprietà che $\text{enc}(m_1, k) \oplus \text{enc}(m_2, k) = \text{enc}(m_1 \oplus m_2, k)$, dove $\oplus$ rappresenta un'operazione (come l'addizione o la moltiplicazione) che può essere eseguita sia sui messaggi in chiaro che sui testi cifrati. Questa proprietà è naturalmente espressa da un'equazione, ma l'equazione non è orientabile in una direzione preferita: non è chiaro se dovrebbe essere applicata da sinistra a destra o da destra a sinistra.

Equazioni non orientabili portano a teorie equazionali più complesse dove possono esistere multiple forme normali per lo stesso termine, o dove la riduzione a forma normale può richiedere di applicare equazioni in entrambe le direzioni. Questo rende la verifica significativamente più difficile, poiché decidere se due termini sono equivalenti può richiedere di esplorare uno spazio di possibili riduzioni, e in generale il problema di equivalenza di termini può diventare indecidibile per teorie equazionali arbitrarie.

Parte V: Computational Soundness e la Riconciliazione dei Modelli

Il Divario tra Modello Simbolico e Modello Computazionale

Per decenni, la ricerca sulla verifica formale dei protocolli di sicurezza è stata divisa tra due approcci fondamentalmente diversi: il modello simbolico, o modello Dolev-Yao, e il modello computazionale. Questa divisione ha creato una tensione produttiva ma anche problematica

nella comunità scientifica, con i ricercatori in verifica formale che preferivano il modello simbolico per la sua trattabilità, e i crittografi che insistevano sul modello computazionale per il suo realismo.

Nel modello simbolico, come abbiamo visto nei process calculi, le operazioni crittografiche sono trattate come funzioni perfette astratte. Un messaggio cifrato con una chiave è completamente opaco senza quella chiave, non esiste nozione di probabilità di successo negli attacchi, e non esistono risorse computazionali limitate. Questo modello è estremamente conveniente per la verifica automatica perché permette di ridurre il problema di sicurezza a questioni combinatoriche e simboliche che possono essere affrontate con tecniche di dimostrazione automatica, model checking, o analisi statica.

Nel modello computazionale, al contrario, le primitive crittografiche sono implementate come algoritmi concreti eseguiti da macchine di Turing probabilistiche polinomialmente limitate. La sicurezza è definita in termini di indistinguibilità computazionale: un sistema è sicuro se ogni avversario efficiente, che esegue in tempo polinomiale, ha solo una probabilità trascurabile, che decresce più velocemente di qualsiasi polinomio inverso, di violare una proprietà di sicurezza. Questo modello è più realistico e aderente alla pratica crittografica moderna, ma rende la verifica estremamente difficile e tipicamente non automatizzabile.

Il divario tra questi due modelli ha creato una situazione insoddisfacente. I ricercatori in verifica formale potevano dimostrare automaticamente che un protocollo è sicuro nel modello simbolico, ma queste dimostrazioni non fornivano garanzie sulla sicurezza del protocollo quando implementato con primitive crittografiche reali. I crittografi potevano dimostrare teoremi di sicurezza nel modello computazionale, ma queste dimostrazioni richiedevano tipicamente mesi o anni di lavoro manuale anche per protocolli relativamente semplici, e erano estremamente suscettibili a errori umani.

La domanda cruciale che emerse negli anni Novanta fu: quando una dimostrazione di sicurezza nel modello simbolico implica sicurezza nel modello computazionale? In altre parole, sotto quali condizioni i due modelli sono suoni l'uno rispetto all'altro? Questa domanda ha motivato un'intera linea di ricerca sulla computational soundness che ha prodotto alcuni dei risultati più profondi e tecnicamente sofisticati nella teoria della sicurezza informatica.

I Primi Risultati di Computational Soundness

Il primo risultato significativo di computational soundness fu ottenuto da Martín Abadi e Phillip Rogaway nel 2000, in un articolo seminale intitolato "Reconciling Two Views of Cryptography". Questo articolo affrontò il caso specifico della cifratura simmetrica, dimostrando che sotto certe condizioni ben definite, la nozione simbolica di segretezza implica la nozione computazionale di indistinguibilità semantica.

Il setting considerato da Abadi e Rogaway è il seguente. Si considera un insieme di messaggi che vengono cifrati usando un numero limitato di chiavi simmetriche. Nel modello simbolico, un avversario può vedere i testi cifrati e può decifrare un testo cifrato se e solo se possiede la chiave corrispondente. La domanda è: se un messaggio è segreto nel modello simbolico, ovvero l'avversario non può dedurlo, è anche computazionalmente sicuro quando si usa uno schema di cifratura simmetrica concreto?

La risposta è affermativa, ma solo sotto condizioni precise sullo schema di cifratura. Abadi e Rogaway introdussero la nozione di cifratura IND-CCA2-sicura con pattern-revealing, ovvero uno schema di cifratura che garantisce indistinguibilità sotto attacchi di chosen-ciphertext adattivi, IND-CCA2, e che rivela la struttura dei messaggi nel senso che se due testi cifrati sono stati creati con la stessa chiave a partire dallo stesso messaggio, questo è verificabile, ma nessun'altra informazione trapela.

Il teorema principale afferma che se si usa uno schema di cifratura con queste proprietà, allora per ogni avversario computazionale polinomialmente limitato che attacca il sistema reale, esiste un avversario simbolico, nel modello Dolev-Yao, che può dedurre le stesse informazioni. Quindi, se un messaggio non può essere dedotto nel modello simbolico, nessun avversario computazionale efficiente può imparare informazioni significative su quel messaggio.

La dimostrazione di questo teorema è tecnicamente sofisticata e introduce diverse idee che sono diventate standard nei risultati successivi di computational soundness. L'idea chiave è costruire una simulazione che mappa ogni attacco computazionale in un attacco simbolico equivalente. Questa simulazione deve gestire la differenza tra le rappresentazioni dei dati nei due modelli: nel modello simbolico i messaggi sono termini simbolici, mentre nel modello computazionale sono stringhe di bit.

La simulazione procede per passi. Dato un avversario computazionale A , si costruisce un avversario simbolico S che simula A nel modello simbolico. Quando A esegue un'operazione crittografica, come cifrare un messaggio o applicare una funzione hash, S esegue la corrispondente operazione simbolica. La sfida è garantire che questa simulazione sia fedele, ovvero che A e S vedano distribuzioni di messaggi indistinguibili.

Per garantire che la simulazione sia fedele, è cruciale che lo schema di cifratura abbia certe proprietà di non-malleabilità e di robustezza rispetto alla composizione. Lo schema deve impedire all'avversario di creare nuovi testi cifrati validi modificando testi cifrati esistenti in modi non banali, e deve garantire che testi cifrati creati indipendentemente rimangano indipendenti anche quando l'avversario può osservare molti testi cifrati.

La condizione IND-CCA2 è essenziale per garantire queste proprietà. Uno schema IND-CCA2-sicuro garantisce che un avversario che ha accesso a un oracolo di decifratura, che può decifrare qualsiasi testo cifrato scelto dall'avversario, eccetto il challenge ciphertext, non può distinguere la cifratura di un messaggio dalla cifratura di un altro messaggio. Questa proprietà è più forte della semplice indistinguibilità semantica, IND-CPA, e cattura l'idea che la cifratura rimane sicura anche in presenza di attacchi attivi sofisticati.

Il risultato di Abadi e Rogaway fu rivoluzionario perché dimostrò per la prima volta che il modello simbolico non è solo una conveniente approssimazione, ma può fornire garanzie formali sulla sicurezza computazionale reale, purché le primitive crittografiche soddisfino proprietà appropriate. Questo risultato ha aperto la strada a una ricca linea di ricerca che ha esteso questi risultati a primitive crittografiche più complesse, a classi più ampie di protocolli, e a proprietà di sicurezza più sofisticate.

Estensioni a Primitive Crittografiche Complesse

Dopo il risultato pionieristico di Abadi e Rogaway sulla cifratura simmetrica, numerosi ricercatori hanno lavorato per estendere i risultati di computational soundness a un'ampia varietà di primitive crittografiche. Ogni primitiva presenta sfide tecniche specifiche che richiedono nuove idee e tecniche di dimostrazione.

La cifratura a chiave pubblica fu uno dei primi obiettivi di estensione. A differenza della cifratura simmetrica, dove la chiave di cifratura deve rimanere segreta, nella cifratura a chiave pubblica la chiave di cifratura è pubblica e quindi disponibile all'avversario. Questo introduce nuove complicazioni: l'avversario può generare arbitrariamente molti testi cifrati usando la chiave pubblica, e deve essere impedito di usare questi testi cifrati per guadagnare informazioni sui messaggi cifrati dal protocollo onesto.

I risultati di computational soundness per la cifratura a chiave pubblica richiedono che lo schema soddisfi IND-CCA2-security proprio come nel caso simmetrico, ma le dimostrazioni sono più delicate perché devono gestire il fatto che l'avversario può generare nuovi testi cifrati. La simulazione deve garantire che testi cifrati generati dall'avversario siano trattati consistentemente nei due modelli, il che richiede tecniche sofisticate per tracciare la provenienza dei testi cifrati e per gestire collisioni accidentali.

Le firme digitali presentano sfide diverse. Una firma è verificabile da chiunque possieda la chiave pubblica, quindi il modello simbolico deve catturare il fatto che l'avversario può verificare firme ma non può forgiare firme valide senza la chiave privata. I risultati di computational soundness per le firme richiedono che lo schema soddisfi sicurezza contro forgery esistenziale sotto attacchi di chosen-message, EUF-CMA, che garantisce che un avversario non può produrre una firma valida su un nuovo messaggio anche avendo visto firme su messaggi scelti dall'avversario.

Un aspetto delicato con le firme è il trattamento della verificabilità pubblica. Nel modello simbolico, una firma su un messaggio può essere modellata come un termine $\text{sign}(m, k)$ dove k è la chiave privata, e la verifica è modellata testando se un termine ha questa forma. Nel modello computazionale, la verifica è un algoritmo probabilistico che può accettare o rifiutare, e devono essere gestiti casi di firme malformate o di messaggi che hanno firme valide per ragioni spurie.

Le funzioni hash crittografiche sono particolarmente problematiche per i risultati di computational soundness. Nel modello simbolico, una funzione hash è tipicamente modellata come una funzione iniettiva senza equazioni, rappresentando l'idea che hash diversi provengono da input diversi e che l'hash non rivela informazioni sull'input. Nel modello computazionale, una funzione hash è una funzione concreta che mappa stringhe arbitrarie a stringhe di lunghezza fissa, e deve soddisfare proprietà come resistenza alle collisioni, resistenza alla preimmagine, e resistenza alla seconda preimmagine.

Il problema è che le funzioni hash computazionali non possono essere iniettive, per ragioni di cardinalità: il dominio è più grande del codominio, quindi possono verificarsi collisioni. Tuttavia, se la funzione hash è collision-resistant, le collisioni sono computazionalmente irreperibili da un avversario efficiente. I risultati di computational soundness devono quindi gestire questo disallineamento: nel modello simbolico le collisioni non esistono, mentre nel modello computazionale esistono ma sono irreperibili.

La soluzione tipica è introdurre una semantica probabilistica che assegna probabilità trascurabili agli eventi di collisione. Si dimostra che se un attacco computazionale ha successo, allora o l'avversario ha trovato una collisione, che accade con probabilità trascurabile se la funzione hash è collision-resistant, oppure esiste un attacco simbolico corrispondente. Quindi, l'assenza di attacchi simbolici implica che gli attacchi computazionali hanno successo solo con probabilità trascurabile.

I Message Authentication Codes, MAC, sono un'altra primitiva importante per cui sono stati ottenuti risultati di computational soundness. Un MAC è simile a una firma ma usa crittografia simmetrica: per generare un MAC su un messaggio serve una chiave segreta, e la stessa chiave serve per verificare il MAC. Nel modello simbolico, un MAC può essere modellato come $\text{mac}(m, k)$, e solo chi possiede k può generare o verificare MAC.

I risultati di computational soundness per i MAC richiedono che lo schema soddisfi sicurezza contro forgery esistenziale, EUF-CMA, analogamente alle firme. Tuttavia, a differenza delle firme, i MAC non sono verificabili pubblicamente, il che introduce nuove complicazioni nella simulazione: bisogna tracciare quali agenti hanno accesso alle chiavi dei MAC e garantire che solo questi agenti possano verificare MAC.

La Composizione di Primitive e Protocolli

Una delle sfide più significative nei risultati di computational soundness è gestire la composizione: quando multiple primitive crittografiche sono usate insieme nello stesso protocollo, o quando multiple istanze dello stesso protocollo vengono eseguite concorrentemente, è garantito che la sicurezza si preserva? Questa questione è cruciale per l'applicabilità pratica dei risultati, poiché i protocolli reali tipicamente compongono multiple primitive e eseguono molte sessioni concorrenti.

Il problema della composizione emerge perché le definizioni di sicurezza standard per le primitive crittografiche sono tipicamente dimostrate in isolamento, senza considerare come la primitiva interagisce con altre primitive o con il contesto più ampio del protocollo. Ad esempio, uno schema di cifratura potrebbe essere IND-CCA2-sicuro quando usato da solo, ma quando composto con uno schema di firma nello stesso protocollo, potrebbero emergere vulnerabilità impreviste.

Un esempio classico di fallimento della composizione è il confronto tra encrypt-then-MAC e MAC-then-encrypt. Se si cifra un messaggio e poi si calcola un MAC sul testo cifrato, encrypt-then-MAC, il protocollo risultante è tipicamente sicuro. Se invece si calcola un MAC sul messaggio in chiaro e poi si cifra tutto insieme, MAC-then-encrypt o encrypt-and-MAC, possono emergere vulnerabilità che permettono attacchi di padding oracle o altre forme di attacchi di chosen-ciphertext.

I risultati di computational soundness devono quindi garantire che la simulazione dal modello computazionale al modello simbolico preservi la correttezza anche quando multiple primitive sono composte. Questo tipicamente richiede condizioni più forti sulle primitive. Ad esempio, invece di richiedere solo IND-CCA2-security per la cifratura, potrebbe essere necessario richiedere anche non-committing encryption, che garantisce che un testo cifrato non rivela informazioni sulla chiave usata per cifrarlo, anche quando l'avversario vede successivamente la chiave.

La nozione di joint state composability è stata introdotta per formalizzare quando le primitive possono essere composte in sicurezza. Una primitiva è jointly state composable se rimane sicura anche quando lo stato interno, come le chiavi o la randomness, è potenzialmente condiviso con altre istanze della stessa primitiva o con altre primitive. Questa proprietà è più forte delle nozioni di sicurezza standard e non è soddisfatta da tutti gli schemi pratici.

Il framework del Universal Composability, UC, introdotto da Ran Canetti, fornisce un approccio generale al problema della composizione. Nel framework UC, ogni protocollo è specificato insieme a una funzionalità ideale che descrive cosa il protocollo dovrebbe realizzare in un mondo ideale dove tutti i canali sono sicuri e tutte le parti sono oneste. Un protocollo è UC-sicuro se emula la funzionalità ideale, nel senso che per ogni attacco nel mondo reale esiste un attacco equivalente nel mondo ideale.

Il teorema di composizione dell'UC garantisce che protocolli UC-sicuri possono essere composti arbitrariamente mantenendo la sicurezza. Se un protocollo di alto livello è dimostrato sicuro assumendo funzionalità ideali, e queste funzionalità sono realizzate da protocolli UC-sicuri, allora il protocollo composto è automaticamente sicuro. Questo permette una progettazione modulare dove ogni componente è verificato indipendentemente e la sicurezza del sistema completo segue automaticamente.

I risultati di computational soundness nel framework UC sono particolarmente potenti. Si può dimostrare che se un protocollo è sicuro nel modello simbolico, secondo certe definizioni di sicurezza simbolica, e se le primitive crittografiche usate realizzano in modo UC-sicuro le corrispondenti funzionalità ideali, allora il protocollo è UC-sicuro nel modello computazionale. Questo fornisce una giustificazione formale per l'uso del modello simbolico nella verifica automatica, anche per protocolli complessi e composti.

Computational Soundness per l'Applied Pi-Calculus

Estendere i risultati di computational soundness all'applied π -calculus è stato un obiettivo importante perché questo framework è usato da molti strumenti di verifica automatica come ProVerif. La sfida principale è che l'applied π -calculus è estremamente espressivo e permette di specificare teorie equazionali arbitrarie, rendendo difficile dare condizioni generali sotto le quali la computational soundness è garantita.

I primi risultati in questa direzione furono ottenuti da Mathieu Baudet e da Véronique Cortier e colleghi, che svilupparono versioni computazionali dell'applied π -calculus dove i termini simbolici sono interpretati come distribuzioni di probabilità su stringhe di bit. In questa semantica computazionale, due termini simbolici sono considerati equivalenti se le corrispondenti distribuzioni sono computazionalmente indistinguibili.

Il teorema di computational soundness afferma che se due processi sono trace equivalent nel modello simbolico, ovvero producono frame indistinguibili nel senso della deducibilità statica, allora sono anche computazionalmente indistinguibili, ovvero producono distribuzioni di messaggi indistinguibili per avversari polinomialmente limitati. Questo risultato giustifica l'uso di strumenti come ProVerif per verificare proprietà di equivalenza: se ProVerif dimostra che due processi sono equivalenti nel modello simbolico, allora sono anche equivalenti nel modello computazionale, sotto appropriate assunzioni sulle primitive crittografiche.

La dimostrazione di questi risultati richiede condizioni tecniche sulle teorie equazionali e sulle implementazioni computazionali delle funzioni. Una condizione tipica è che la teoria equazionale sia ben strutturata nel senso che non permetta di derivare equazioni spurie che non corrispondono a proprietà reali delle primitive computazionali. Ad esempio, se la teoria equazionale include un'equazione $f(g(x)) = x$ che dovrebbe rappresentare l'inversione di una funzione, allora l'implementazione computazionale di g deve essere effettivamente invertibile tramite f .

Un aspetto delicato è il trattamento delle equazioni che rappresentano proprietà algebriche delle primitive. Ad esempio, la cifratura omomorfica ha equazioni come $\text{enc}(m_1, k) \oplus \text{enc}(m_2, k) = \text{enc}(m_1 \oplus m_2, k)$. Per ottenere computational soundness, l'implementazione computazionale della cifratura deve soddisfare queste equazioni non solo esattamente, ma anche in modo robusto nel senso che piccole perturbazioni nei testi cifrati non permettono all'avversario di distinguere distribuzioni che dovrebbero essere indistinguibili.

I risultati di computational soundness per l'applied π -calculus hanno anche affrontato la questione della gestione dello stato e della randomness. Nel modello simbolico, le operazioni crittografiche sono deterministiche: cifrare lo stesso messaggio con la stessa chiave produce sempre lo stesso testo cifrato. Nel modello computazionale, le operazioni sono tipicamente probabilistiche: cifrare lo stesso messaggio due volte produce testi cifrati diversi. Questa differenza deve essere gestita nella simulazione, tipicamente assumendo che il modello simbolico rappresenti implicitamente la randomness attraverso nonce freschi.

Un risultato importante di Cortier, Kremer e Warinschi ha dimostrato computational soundness per una classe ampia di teorie equazionali che include la maggior parte delle primitive standard, cifratura simmetrica e asimmetrica, firme, MAC, funzioni hash, insieme a proprietà algebriche comuni come cifratura omomorfica, blind signatures, commitment schemes. Questo risultato fornisce fondamenti teorici per l'uso di ProVerif e strumenti simili nella verifica di protocolli reali, con la garanzia che le proprietà verificate nel modello simbolico si trasferiscono al modello computazionale.

Prospettive Future e Direzioni Aperte

La ricerca sulle logiche per la sicurezza e sulla computational soundness continua a evolversi, affrontando nuove sfide poste dall'emergere di nuove tecnologie e paradigmi crittografici. Diverse direzioni di ricerca aperte promettono sviluppi significativi nei prossimi anni.

La crittografia post-quantistica richiede lo sviluppo di nuovi risultati di computational soundness per primitive basate su problemi matematici diversi da quelli tradizionali. Le proprietà algebriche di schemi basati su reticoli, su codici, o su isogenie sono significativamente diverse da quelle della crittografia classica, e richiedono nuove teorie equazionali e nuove tecniche di dimostrazione.

La verifica di protocolli che usano tecnologie blockchain e smart contracts presenta sfide uniche legate alla gestione dello stato globale condiviso, alla programmabilità delle condizioni di sicurezza, e alla composizione di protocolli eseguiti da attori non fidati. Gli approcci tradizionali basati su process calculi devono essere estesi per gestire la semantica dei linguaggi di smart contract e le garanzie fornite dal consenso distribuito.

L'integrazione di garanzie di privacy differenziale nei framework di verifica formale è un'altra direzione promettente. La privacy differenziale fornisce garanzie quantitative sulla leakage di informazione, complementari alle garanzie qualitative tipicamente fornite dalle logiche per la sicurezza. Sviluppare un framework unificato che combini verifica simbolica e analisi quantitativa della privacy è un obiettivo ambizioso ma importante.

La verifica di protocolli in presenza di side channels e di implementazioni concrete rimane una sfida aperta. I risultati di computational soundness tipicamente assumono che le primitive crittografiche siano implementate correttamente e che non esistano canali laterali. Estendere questi risultati per gestire timing attacks, power analysis, e altre forme di leakage implementation-level richiede collegare i modelli simbolici non solo con la crittografia teorica ma anche con l'analisi di sicurezza a livello di codice e hardware.

Nonostante queste sfide aperte, i progressi compiuti negli ultimi decenni hanno trasformato la verifica dei protocolli di sicurezza da un'arte oscura praticata da pochi esperti in una disciplina scientifica rigorosa con solide fondamenta teoriche e strumenti pratici efficaci. Le logiche specifiche per la sicurezza, con tutte le loro varianti ed estensioni, rappresentano uno dei successi più significativi dell'applicazione di metodi formali a problemi di sicurezza del mondo reale. Cifratura simmetrica, dove la chiave di cifratura deve rimanere segreta, nella cifratura a chiave pubblica la chiave di cifratura è pubblica e quindi disponibile all'avversario. Questo introduce nuove complicazioni: l'avversario può generare arbitrariamente molti testi cifrati usando la chiave pubblica, e deve essere impedito di usare questi testi cifrati per guadagnare informazioni sui messaggi cifrati dal protocollo onesto. I risultati di computational soundness per la cifratura a chiave pubblica richiedono che lo schema soddisfi IND-CCA2-security proprio come nel caso simmetrico, ma le dimostrazioni sono più delicate perché devono gestire il fatto che l'avversario può generare nuovi testi cifrati. La simulazione deve garantire che testi cifrati generati dall'avversario siano trattati consistentemente nei due modelli, il che richiede tecniche sofisticate per tracciare la provenienza dei testi cifrati e per gestire collisioni accidentali. Le firme digitali presentano sfide diverse. Una firma è verificabile da chiunque possieda la chiave pubblica, quindi il modello simbolico deve catturare il fatto che l'avversario può verificare firme ma non può forgiare firme valide senza la chiave privata. I risultati di computational soundness per le firme richiedono che lo schema soddisfi sicurezza contro forgery esistenziale sotto attacchi di chosen-message (EUF-CMA), che garantisce che un avversario non può produrre una firma valida su un nuovo messaggio anche avendo visto firme su messaggi scelti dall'avversario.

Un aspetto delicato con le firme è il trattamento della verificabilità pubblica. Nel modello simbolico, una firma su un messaggio può essere modellata come un termine $\text{sign}(m, k)$ dove k è la chiave privata, e la verifica è modellata testando se un termine ha questa forma. Nel modello computazionale, la verifica è un algoritmo probabilistico che può accettare o rifiutare, e devono essere gestiti casi di firme malformate o di messaggi che hanno firme valide per ragioni spurie.

Le funzioni hash crittografiche sono particolarmente problematiche per i risultati di computational soundness. Nel modello simbolico, una funzione hash è tipicamente modellata come una funzione iniettiva senza equazioni, rappresentando l'idea che hash diversi provengono da input diversi e che l'hash non rivela informazioni sull'input. Nel

modello computazionale, una funzione hash è una funzione concreta che mappa stringhe arbitrarie a stringhe di lunghezza fissa, e deve soddisfare proprietà come resistenza alle collisioni, resistenza alla preimmagine, e resistenza alla seconda preimmagine.

Il problema è che le funzioni hash computazionali non possono essere iniettive (per ragioni di cardinalità: il dominio è più grande del codominio), quindi possono verificarsi collisioni. Tuttavia, se la funzione hash è collision-resistant, le collisioni sono computazionalmente irreperibili da un avversario efficiente. I risultati di computational soundness devono quindi gestire questo disallineamento: nel modello simbolico le collisioni non esistono, mentre nel modello computazionale esistono ma sono irreperibili.

La soluzione tipica è introdurre una "semantica probabilistica" che assegna probabilità trascurabili agli eventi di collisione. Si dimostra che se un attacco computazionale ha successo, allora o l'avversario ha trovato una collisione (che accade con probabilità trascurabile se la funzione hash è collision-resistant), oppure esiste un attacco simbolico corrispondente. Quindi, l'assenza di attacchi simbolici implica che gli attacchi computazionali hanno successo solo con probabilità trascurabile.

I Message Authentication Codes (MAC) sono un'altra primitiva importante per cui sono stati ottenuti risultati di computational soundness. Un MAC è simile a una firma ma usa crittografia simmetrica: per generare un MAC su un messaggio serve una chiave segreta, e la stessa chiave serve per verificare il MAC. Nel modello simbolico, un MAC può essere modellato come $\text{mac}(m, k)$, e solo chi possiede k può generare o verificare MAC.

I risultati di computational soundness per i MAC richiedono che lo schema soddisfi sicurezza contro forgery esistenziale (EUF-CMA), analogamente alle firme. Tuttavia, a differenza delle firme, i MAC non sono verificabili pubblicamente, il che introduce nuove complicazioni nella simulazione: bisogna tracciare quali agenti hanno accesso alle chiavi dei MAC e garantire che solo questi agenti possano verificare MAC.

La Composizione di Primitive e Protocolli

Una delle sfide più significative nei risultati di computational soundness è gestire la composizione: quando multiple primitive crittografiche sono usate insieme nello stesso protocollo, o quando multiple istanze dello stesso protocollo vengono eseguite concorrentemente, è garantito che la sicurezza si preserva? Questa questione è cruciale per l'applicabilità pratica dei risultati, poiché i protocolli reali tipicamente compongono multiple primitive e eseguono molte sessioni concorrenti.

Il problema della composizione emerge perché le definizioni di sicurezza standard per le primitive crittografiche sono tipicamente dimostrate in isolamento, senza considerare come la primitiva interagisce con altre primitive o con il contesto più ampio del protocollo. Ad esempio, uno schema di cifratura potrebbe essere IND-CCA2-sicuro quando usato da solo, ma quando composto con uno schema di firma nello stesso protocollo, potrebbero emergere vulnerabilità impreviste.

Un esempio classico di fallimento della composizione è il "commit-then-encrypt" versus "encrypt-then-MAC". Se si cifra un messaggio e poi si calcola un MAC sul testo cifrato (encrypt-then-MAC), il protocollo risultante è tipicamente sicuro. Se invece si calcola un

MAC sul messaggio in chiaro e poi si cifra tutto insieme (MAC-then-encrypt o encrypt-and-MAC), possono emergere vulnerabilità che permettono attacchi di padding oracle o altre forme di attacchi di chosen-ciphertext.

I risultati di computational soundness devono quindi garantire che la simulazione dal modello computazionale al modello simbolico preservi la correttezza anche quando multiple primitive sono composte. Questo tipicamente richiede condizioni più forti sulle primitive. Ad esempio, invece di richiedere solo IND-CCA2-security per la cifratura, potrebbe essere necessario richiedere anche non-committing encryption, che garantisce che un testo cifrato non rivela informazioni sulla chiave usata per cifrarlo, anche quando l'avversario vede successivamente la chiave.

La nozione di joint state composability è stata introdotta per formalizzare quando le primitive possono essere composte in sicurezza. Una primitiva è jointly state composable se rimane sicura anche quando lo stato interno (come le chiavi o la randomness) è potenzialmente condiviso con altre istanze della stessa primitiva o con altre primitive. Questa proprietà è più forte delle nozioni di sicurezza standard e non è soddisfatta da tutti gli schemi pratici.

Il framework del Universal Composability (UC), introdotto da Ran Canetti, fornisce un approccio generale al problema della composizione. Nel framework UC, ogni protocollo è specificato insieme a una "funzionalità ideale" che descrive cosa il protocollo dovrebbe realizzare in un mondo ideale dove tutti i canali sono sicuri e tutte le parti sono oneste. Un protocollo è UC-sicuro se "emula" la funzionalità ideale, nel senso che per ogni attacco nel mondo reale esiste un attacco equivalente nel mondo ideale.

Il teorema di composizione dell'UC garantisce che protocolli UC-sicuri possono essere composti arbitrariamente mantenendo la sicurezza. Se un protocollo di alto livello è dimostrato sicuro assumendo funzionalità ideali, e queste funzionalità sono realizzate da protocolli UC-sicuri, allora il protocollo composto è automaticamente sicuro. Questo permette una progettazione modulare dove ogni componente è verificato indipendentemente e la sicurezza del sistema completo segue automaticamente.

I risultati di computational soundness nel framework UC sono particolarmente potenti. Si può dimostrare che se un protocollo è sicuro nel modello simbolico (secondo certe definizioni di sicurezza simbolica), e se le primitive crittografiche usate realizzano in modo UC-sicuro le corrispondenti funzionalità ideali, allora il protocollo è UC-sicuro nel modello computazionale. Questo fornisce una giustificazione formale per l'uso del modello simbolico nella verifica automatica, anche per protocolli complessi e composti.

Computational Soundness per l'Applied Pi-Calculus

Estendere i risultati di computational soundness all'applied π -calculus è stato un obiettivo importante perché questo framework è usato da molti strumenti di verifica automatica come ProVerif. Il challenge principale è che l'applied π -calculus è estremamente espressivo e permette di specificare teorie equazionali arbitrarie, rendendo difficile dare condizioni generali sotto le quali la computational soundness è garantita.

I primi risultati in questa direzione furono ottenuti da Mathieu Baudet e da Véronique Cortier e colleghi, che svilupparono versioni computazionali dell'applied π -calculus dove i termini

simbolici sono interpretati come distribuzioni di probabilità su stringhe di bit. In questa semantica computazionale, due termini simbolici sono considerati equivalenti se le corrispondenti distribuzioni sono computazionalmente indistinguibili.

Il teorema di computational soundness afferma che se due processi sono trace equivalent nel modello simbolico (ovvero producono frame indistinguibili nel senso della deducibilità statica), allora sono anche computazionalmente indistinguibili (ovvero producono distribuzioni di messaggi indistinguibili per avversari polinomialmente limitati). Questo risultato giustifica l'uso di strumenti come ProVerif per verificare proprietà di equivalenza: se ProVerif dimostra che due processi sono equivalenti nel modello simbolico, allora sono anche equivalenti nel modello computazionale (sotto appropriate assunzioni sulle primitive crittografiche). La dimostrazione di questi risultati richiede condizioni tecniche sulle teorie equazionali e sulle implementazioni computazionali delle funzioni. Una condizione tipica è che la teoria equazionale sia "ben strutturata" nel senso che non permetta di derivare equazioni spurie che non corrispondono a proprietà reali delle primitive computazionali. Ad esempio, se la teoria equazionale include un'equazione $f(g(x)) = x$ che dovrebbe rappresentare l'inversione di una funzione, allora l'implementazione computazionale di g deve essere effettivamente invertibile tramite f . Un aspetto delicato è il trattamento delle equazioni che rappresentano proprietà algebriche delle primitive. Ad esempio, la cifratura omomorfa ha equazioni come $\text{enc}(m_1, k) \oplus \text{enc}(m_2, k) = \text{enc}(m_1 \oplus m_2, k)$. Per ottenere computational soundness, l'implementazione computazionale della cifratura deve soddisfare queste equazioni non solo esattamente, ma anche in modo "robusto" nel senso che piccole perturbazioni nei testi cifrati non permettono all'avversario di distinguere distribuzioni che dovrebbero essere indistinguibili. I risultati di computational soundness per l'applied π -calculus hanno anche affrontato la questione della gestione dello stato e della randomness. Nel modello simbolico, le operazioni crittografiche sono deterministiche: cifrare lo stesso messaggio con la stessa chiave produce sempre lo stesso testo cifrato. Nel modello computazionale, le operazioni sono tipicamente probabilistiche: cifrare lo stesso messaggio due volte produce testi cifrati diversi. Questa differenza deve essere gestita nella simulazione, tipicamente assumendo che il modello simbolico rappresenta implicitamente la randomness attraverso nonce freschi. Un risultato importante di Cortier, Kremer e Warinschi ha dimostrato computational soundness per una classe ampia di teorie equazionali che include la maggior parte delle primitive standard (cifratura simmetrica e asimmetrica, firme, MAC, funzioni hash) insieme a proprietà algebriche comuni (cifratura omomorfa, blind signatures, commitment schemes). Questo risultato fornisce fondamenti teorici per l'uso di ProVerif e strumenti simili nella verifica di protocolli reali, con la garanzia che le proprietà verificate nel modello simbolico si trasferiscono al modello computazionale.